Infraestructura de comunicación e interacción para el sector turístico

David Ángel Mata Lorenzo
Delfín Álvaro Miguel Gómez
Enrique Ituarte Martínez-Millán
Rodrigo Notario Pérez

DEPARTAMENTO DE INGENIERÍA DEL SOFTWARE E INTELIGENCIA ARTIFICIAL
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO FIN DE GRADO

JUNIO 2017

Director: Juan Antonio Recio García
Colaborador: Felipe Santi (Sismotur)

AUTORIZACIÓN PARA LA DIFUSIÓN DEL TRABAJO FIN DE GRADO Y SU DEPÓSITO EN EL REPOSITORIO INSTITUCIONAL E-PRINTS COMPLUTENSE

Los abajo firmantes, alumno/s y tutor/es del Trabajo Fin de Grado (TFG) en el Grado en …… de la Facultad de ……, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Grado (TFG) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

Periodo de embargo (opcional):

- ☐ 6 meses
- ☐ 12 meses

TÍTULO del TFG:

……

Curso académico: ….. / …..

Nombre del Alumno/s:

……

……

Tutor/es del TFG y departamento al que pertenece:

……

……

Firma del alumno/s Firma del tutor/es

Agradecimientos

A continuación, vamos a aprovechar estas líneas para agradecer a todas aquellas personas que nos han acompañado y nos han ayudado durante todos estos años de carrera.

En primer lugar, queremos dar nuestro más sentido agradecimiento a la persona que ha dirigido nuestro trabajo de fin de grado, Juan Antonio Recio García. Te agradecemos habernos dado la posibilidad de realizar este proyecto y, sobre todo, queremos agradecerte el tiempo dedicado y la atención recibida, ya que ante cualquier problema siempre nos ofrecías una solución.

También nos gustaría dar un gran agradecimiento a Felipe Santi, nuestro codirector y a su vez CTO de la empresa *Sismotur*, para la cual hemos realizado este trabajo. Gracias por todo lo que nos has enseñado, ya que aparte de explicarnos cosas relacionadas con el proyecto, siempre ibas más allá y aprovechabas las reuniones semanales o quincenales para enseñarnos cómo funcionaban las cosas y cómo desenvolvernos en la vida real.

Además, queremos agradecerles a todos nuestros amigos esos buenos momentos que hemos vivido durante la carrera y que esperamos que sigamos viviendo juntos.

Y, por último, no nos queremos olvidar de mostrar nuestro agradecimiento a todos los profesores de la FDI, los cuales nos habéis aguantado durante todos estos años y sin vosotros no habría sido posible llegar hasta aquí.

Índice

Índice de figuras

Resumen

Vivimos en un mundo totalmente señalizado y marcado por el uso de las tecnologías móviles, donde el sector turístico se aventaja de múltiples avances como la conexión a Internet, la Nube, la geolocalización, la señalización inteligente y diversos métodos publicitarios que atraen a los viajeros. Todos estos servicios tratan de ofrecer una mejor experiencia al turista, ahorrándole una gran carga de trabajo a la hora de decidir qué tipo de actividades realizar, dónde alojarse o en qué restaurantes merece la pena comer, además de proporcionar información sobre los puntos de interés más comunes de la zona turística.

Con todo esto, el viajante tiene un amplio abanico de posibilidades a la hora de planear un viaje. Escogerá un hotel donde hospedarse y en unos días habrá vivido una experiencia inolvidable. Sin embargo, hay un factor poco desarrollado en el mundo del turismo: un sistema de comunicación transparente que permita a la persona interactuar con el proveedor de los servicios a los que se ha suscrito.

En este Trabajo de Fin de Grado, se ha realizado una colaboración con la empresa turística *Sismotur*, propulsora de un nuevo sistema de información turística digital por medio del uso de señalización inteligente, llamado *Inventrip*, que se apoya en tecnologías novedosas como *beacons* basados en *Bluetooth* que han sido usadas por compañías como *Google* y *Apple*. El proyecto abarca la realización de un chat para *Inventrip*, cubriendo diversos casos de uso concretos y útiles tanto para el turista en su aplicación móvil como para el usuario administrador, ya sea la recepción de un hotel o un guía turístico.

***Lista de palabras clave**: Sismotur, Inventrip, beacons, API, SendBird, Chat SDK, Firebase.*

Abstract

We live in a world utterly signalized and touched by the use of mobile device technologies, where the sector of tourism takes advantage of multiple leaps forward such as Internet connectivity, the Cloud, geolocation, smart signaling and many marketing strategies which attract travelers. All these services try to offer a better experience for the tourist, saving up on a great deal of tasks when it comes to deciding on the type of activities to embark on, where to accommodate or restaurants worth eating at, besides providing useful information on the most interesting places of interest.

All this in mind, travelers have a wide fan of possibilities when planning a trip. They will choose a hotel to stay and within a few days they will have lived an unforgettable experience. Nevertheless, there is a not very much developed factor in the world of tourism: a transparent communication system which allows the person to interact with the service provider he has signed for.

In this Final Degree Project, the group has collaborated with consulting business company *Sismotur*, propeller of a new digital information system using smart signaling, called *Inventrip*, which supports new coming technologies like *Bluetooth beacons* that have been used by companies such as *Google* and *Apple*. The project covers the implementation of a chat system for *Inventrip*, touching on many use cases, useful for either tourist on his mobile application or even a hotel receptionist or a guide.

***Keywords**: Sismotur, Inventrip, beacons, API, SendBird, Chat SDK, Firebase.*

Capítulo I

MOTIVACIÓN Y OBJETIVOS

Motivación

Imaginemos que somos una familia compuesta por dos matrimonios con respectivos hijos. Han llegado las vacaciones y decidimos planificar un viaje a Ribera de Duero de forma sencilla gracias a la aplicación *Inventrip*.

El hotel seleccionado para hospedarnos es el Hotel Spa Arzuaga. Llegamos a nuestro destino por la tarde, recibimos nuestras tarjetas de las habitaciones y en Recepción nos informan de que en el hotel se ha integrado la tecnología de señalización inteligente por medio de *beacons*, que nos conectará a una red para obtener servicios, como un canal de comunicación directo con el hotel, además de una forma anónima de hablar en canales públicos con el resto de clientes.

El canal de comunicación tiene forma de chat en la aplicación *Inventrip*. Cada habitación tiene un chat con la Recepción, pero nos gustaría poder comunicarnos entre los dos matrimonios y subscribirnos a servicios de forma conjunta, como puede ser el spa. Acorde a nuestros gustos, el hotel nos ofrece esta posibilidad.

Nos despertamos al día siguiente, pero nos despertamos un poco tarde. Gracias al chat, podemos pedir el servicio de desayuno en la habitación, dado que ya han cerrado el comedor. Escribimos un mensaje y en cuestión de minutos podemos degustar de nuestro desayuno.

Además, nos llegan varias recomendaciones, entre las que se incluyen la posibilidad de suscribirse a un servicio de masajes en el spa, y varias excursiones a los viñedos. Ambas nos llaman la atención, por lo que decidimos preguntar a Recepción si se nos puede incluir en los dos servicios.

Objetivos

El objetivo principal es desarrollar mediante tecnología móvil *Android* y tecnologías Web como *JavaScript* el sistema de chat de *Inventrip*, con una interfaz intuitiva y minimalista con el fin de fortalecer la experiencia del turista. Un desglose más claro de los objetivos es el siguiente:

- Fomentar la comunicación bidireccional del usuario con el servicio.
- Facilitar el envío de mensajes y peticiones del turista en forma de texto o imágenes.
- Proporcionar un sistema de creación de chats de habitación al administrador.
- Crear un sistema cómodo para que el administrador pueda gestionar las conversaciones de los clientes en sus canales de habitación, aprovechando el espacio de la ventana del navegador.

- Ofrecer la posibilidad de modificar, añadir, y eliminar usuarios de cualquier chat de habitación.

Como objetivo grupal, se trata de obtener la experiencia de trabajo junto a una empresa real, en el que se aprenderá cómo se gestionan los desarrollos tecnológicos, cómo se toman decisiones en torno a los requisitos de la empresa y en general poner en práctica el ámbito de trabajo empresarial.

Estructura de la memoria

El contenido de la memoria se encuentra organizado de la siguiente manera:

- En este primer capítulo se ha incluido un breve resumen de lo que consiste el Trabajo de Fin de Grado, así como las motivaciones y objetivos del mismo.
- En el segundo capítulo se hablará de *Sismotur*, la empresa para la cual se ha realizado el sistema de chat. Se comentarán aspectos como su historia, su filosofía, sus tecnologías novedosas y el alcance de *Inventrip*.
- En el tercer capítulo se hará una evaluación de los proyectos similares, tanto para Android como en tecnologías Web. Se explicarán las tecnologías utilizadas a lo largo del proyecto, incluyendo una breve comparación y unas conclusiones.
- En el cuarto capítulo, el más extenso, se hablará en detalle del desarrollo del proyecto.
    - Se describirá la metodología de trabajo seguida durante toda la duración de la creación del chat.
    - Se aportará información sobre el factor de forma, la postura, los métodos de entrada y los elementos de datos de la aplicación.
    - Se incluirá una descripción de requisitos funcionales para el chat de *Inventrip*. Estos serán todos los casos de uso implementados en la versión final.
    - Se aportarán detalles sobre el diseño y la implementación de la aplicación tanto en Android como en JavaScript, con las APIs utilizadas en el proceso.
- En el quinto capítulo se realizará una evaluación comparativa desde el punto de vista de la empresa.
    - Se hará una descripción de los problemas más importantes encontrados.
    - Se explicará qué caminos se han seguido para solucionar dichos problemas.
    - Se analizará la viabilidad de las tecnologías utilizadas.
- El sexto capítulo discutirá las conclusiones finales del proyecto.
    - Un breve resumen final.
    - Una vista hacia el trabajo futuro con posibles mejoras.
    - Una descripción de la aportación personal de cada miembro del equipo.

- El séptimo y último capítulo ofrecerá ejemplos de las estructuras de las bases de datos.

Capítulo II

SISMOTUR

Actividad empresarial

Sismotur es una empresa de turismo que nació en el año 2000 como una empresa de consultoría destinada al desarrollo e implementación de sistemas avanzados de señalización e información turística para la promoción de destinos turísticos inteligentes. Han acompañado a más de cien destinos turísticos en la mejora de su promoción e información turística. [1]

Sismotur fue la empresa escogida para la implantación de un servicio llamado *Cloud Signing*, que permitió mediante un sistema de geolocalización poder planificar itinerarios turísticos. Al poco tiempo, se descubrió el potencial de este sistema y se decidió aprovechar las tecnologías actuales para mejorar la experiencia del turista.

Es así como en 2014 nació *Inventrip*, una aplicación web que permite al turista planificar un viaje de forma sencilla y rápida para después compartirlo en las redes sociales. Este proyecto recibió el premio finalista en el V Encuentro de Territorio y Marketing.

Sismotur siempre ha apostado por integrar y atender las necesidades de todos los actores implicados en el sector turístico.

- Los turistas esperan un destino que los acoja y acompañe, dándoles información antes del viaje, recibiéndolos y guiándolos durante su estancia, y escuchándoles una vez terminada la misma.
- Los servicios turísticos necesitan rentabilizar las inversiones realizadas para crear productos y servicios turísticos, y entre otras cosas requieren para ello herramientas de promoción que les den visibilidad para llegar a sus clientes de la manera más directa y económica posible.
- Los destinos desean promover la economía de su territorio de forma sostenible y competitiva, y también aprovechar el turismo para generar riqueza y empleo no deslocalizable.

Sin embargo, los turistas actuales son muy diferentes. Están conectados permanentemente a Internet gracias a nuevas tecnologías como el s*martphone* y la tableta y disponen cada vez de más información sobre los destinos y servicios turísticos. También piden experiencias particulares personalizadas que se adapten a sus gustos y estilo de vida.

Actualmente, los turistas utilizan mayoritariamente los motores de búsqueda para encontrar información turística. Sin embargo, el resultado no siempre responde a sus necesidades, ya que la información suele ser poco precisa, fragmentada y de calidad variable. Por esta razón, Sismotur tiene como objetivo disponer de un sistema integrado que de servicio a todas las partes implicadas en el turismo.

Ámbito tecnológico

Sismotur afirma que el pilar básico de un destino turístico inteligente es el sistema de información digital de sus recursos y servicios. Considera relevante para el sistema las siguientes características:

- El modelo de datos ha de ser único, flexible y dinámico, y debe permitir la caracterización de cualquier recurso o servicio turístico con un nivel de complejidad suficiente, que responda a las necesidades particulares de cada tipo de turismo.
- El lenguaje debe ser común y abierto para que el significado de los datos sea comprensible por todos. Propone, en particular, la adopción de la nomenclatura *Schema.org*, cuyo uso está apoyado por los principales motores de búsqueda.
- Para responder a las necesidades de integración de los distintos actores, Sismotur apoya una estrategia *open data* implementada con una API RESFTUL que publique datos en formatos estándar. Esto permitirá la integración de los datos en tiempo real con otras aplicaciones en la web, como páginas turísticas, motores de reservas, redes sociales, o *apps*.

Sismotur cree que una aplicación importante de Internet de las Cosas (IoT) en el ámbito del turismo será la capacidad de integrar en la señalización interpretativa (tótems, paneles, placas, etc) conexiones al sistema de información turística *online* del destino. Con ello se suma la universalidad de acceso del sistema de señalización a las posibilidades brindadas por las nuevas tecnologías *online*.

La definición de un modelo off/on debe considerar toda la complejidad que interviene en el proceso de diagnóstico, definición, diseño y actualización de los sistemas de señalización e información turística. Un modelo integral se sustenta en cuatro pilares básicos:

- Marco metodológico: análisis y diagnóstico de la situación actual, definición de una política de señalización e información turística, así como de un manual de señalización turística.
- Concertación público-privada: creación de grupos de trabajo transversales y multisectoriales para evitar la descoordinación entre Administraciones e integrar a las empresas turísticas.
- Uso de tecnologías especializadas: se requieren sistemas capaces de trabajar con información georreferenciada, que funcionen en el *cloud computing* y que dispongan de motores de inteligencia artificial social para analizar las preferencias individuales y de grupo.
- Conexiones off/on disponibles: códigos QR, chips NFC y redes de *beacons* o dispositivos análogos.

Beacons

Una vez se implantaron las señalizaciones turísticas en las rutas y tras la creación de *Inventrip*, se propuso utilizar la tecnología de *beacons Eddystone* y se recogió en el Libro Blanco de Destinos Turísticos Inteligentes de SEGITTUR y se presentó en el stand de SEGITTUR en FITUR 2016.

Sismotur optó por usar los beacons de la compañía llamada Kontakt, conocida por ser el líder mundial de esta tecnología. Actualmente se está planteando su uso en paralelo con la tecnología *Nearby*.

Los beacons han de ser configurados usando el software proporcionado por la marca.

Para proporcionar datos, se pueden usar los tipos Eddystone-UID y Eddystone-EID. Para proporcionar enlaces URL se puede usar el tipo Eddystone-URL.

Algunas de las características importantes de estas tecnologías son:

- Permiten la monitorización de los beacons, usando Google Beacon Dashboard.
- Se pueden asociar varias URL o datos a cada beacon.
- Se pueden actualizar los beacons remotamente.
- Se puede controlar quién accede a los beacons. Para más información, consultar Eddystone Ephemeral Identifier.

Google Nearby

Nearby es una tecnología proporcionada por Google destinada a la recepción de notificaciones de servicios en un radio de treinta metros. Una persona puede interactuar con estos servicios desde su teléfono móvil, lo que lo hace un sistema interesante para *Inventrip*.

La forma en la que *Nearby* interactúa con los servicios es por medio de tecnología Bluetooth, WiFi y sonido inaudible. Las notificaciones son diferentes a las habituales, ya que no emiten vibración, ni sonido, aparecen en segundo plano y desaparecen automáticamente. Además, *Nearby* no guarda ni envía ningún tipo de información del dispositivo a terceros.

Puede darse el caso de que un turista se encuentre cerca de uno de los beacons de *Inventrip*. Ésta persona recibiría por medio de *Nearby* una notificación de la posibilidad de instalar la aplicación en su dispositivo, describiendo las funcionalidades principales de la misma.

Si el turista dispone de conectividad a Internet, *Nearby* podría redirigirle a Play Store y a la página web informativa. Si, por lo contrario, no puede conectarse ya que no tiene tarifa

de datos operativa, podría existir la posibilidad de guardarle un recordatorio en el teléfono para cuando disponga de conexión a Internet vía WiFi o similares.

Como requisito principal, el usuario debe tener activada la opción *Nearby* en su dispositivo. Esta opción requiere el uso de Bluetooth de bajo consumo para así poder detectar las emisiones de notificaciones.

Actualmente existen dos maneras distintas de enviar notificaciones *Nearby* a los dispositivos móviles:

- Dar un enlace HTTPS mediante una URL. Cuando el usuario toca la notificación, se le abre una ventana de navegador.
- Disparar un proceso de intento de apertura de la aplicación Inventrip y realizar una acción específica. Si el usuario no tiene descargada la aplicación, se le redirige a Play Store a la dirección exacta de descarga. Esta sería la opción más adecuada para Inventrip, compaginada con algún tipo de página informativa.

Inventrip

Inventrip es un innovador servicio de información turística personalizado que permite consultar la oferta turística de un destino, construir viajes a medida, acceder a los mismos a partir de cualquier dispositivo, y compartirlos por las redes sociales o mediante el uso de dispositivos físicos de conexión (QR, NFC, Beacons).

Esta plataforma recibió en 2014 un premio nacional en España de marketing territorial online por su implantación en la Ruta Ribera del Duero. En 2015 ha lanzado un servicio de vídeos promocionales hechos con drones, *DronTrip*, integrados a los viajes de Inventrip.

Tras implantar la señalización turística en la ruta (canal offline) e *Inventrip* (canal online) se plantea por primera vez una nueva visión para conectar un territorio con el turista digital desplegando el Internet de las Cosas (IoT) utilizando la tecnología *Beacons*. Esta visión *off-on* fue recogida en octubre de 2015 en el Libro Blanco de Destinos Turísticos Inteligentes de SEGITTUR, y presentada en el stand de SEGITTUR en FITUR 2016.

Implementación

La señalización turística inteligente conecta un sistema de información físico (la señal) con el sistema de información del smartphone del turista utilizando la tecnología Beacon a través de Inventrip. Los Beacons son dispositivos electrónicos basados en una novedosa tecnología de Bluetooth de baja energía (BLE) que ayudan a los terminales móviles a entender su emplazamiento y los recursos y servicios que los rodean con un grado de precisión elevado, permitiendo con ello la creación de un territorio inteligente conectado.

El beacon se aloja en el tapón de un poste de una señal o panel de información. En la ilustración adjunta puede verse el detalle del alojamiento del beacons en el panel de información.

Los turistas utilizan el sistema *Inventrip*, cuya interfaz en este caso es una App (necesaria para conectarse por Bluetooth con el *beacon*). *Inventrip* detecta el *beacon* y utiliza su distintivo único para recibir información con los lugares cercanos, consolidando toda la información en un único canal de comunicación con el destino. *Inventrip* ha sido diseñado para tener una interfaz extremadamente sencilla y para responder a preguntas comunes de los visitantes del destino cómo *¿Dónde estoy?, ¿Qué estoy viendo?*, o *¿Qué puedo hacer en este lugar?* Asimismo, la aplicación hace sencilla la visita del destino ya que da acceso a viajes e itinerarios actualizados por el destino

En definitiva, *Sismotur* ha creado una red global de señalización turística conectada basada en tecnología *Beacons* y gestionada por la plataforma Inventrip. La primera implantación ha sido en la Ruta del Vino Ribera del Duero, creando con ello un canal de información offline (señalización turística) y online (conexión al smartphone del visitante) donde cada señal funciona como una oficina de turismo digital. A través de *Inventrip* se conecta el territorio con el turista digita.

La Ribera del Duero

La Ribera del Duero constituye un caso de éxito en la integración de los sistemas de señalización e información turística. La señalización participa no solamente como guía de los recursos, sino que además aporta un valor añadido e imagen al destino. [2]

Para cada uno de los beacons de las 103 señales inteligentes instaladas la Ruta ha definido la información siguiente:

- Cerca de ti: recursos o servicios que se encuentren en la proximidad del punto de instalación.
- Explora a tu alrededor: qué ver, qué hacer, dónde comer y dónde alojarse en el entorno del núcleo donde se encuentre la señal inteligente.
- Propuesta de un viaje Inventrip: por ejemplo, con propuestas de Rutas, senderos o recorridos culturales en el entorno.

Ventajas

- La señalización turística inteligente viene a resolver las limitaciones del actual sistema de señalización físico: problemas de actualización de contenidos de información, imposibilidad de incluir muchos idiomas en los soportes físicos, y la dificultad en recoger la gran oferta de servicios turísticos.

- *Inventrip* es una plataforma muy sencilla de utilizar y resuelve las necesidades de distintos tipos de turistas: aquellos a quienes les gusta planificar su viaje, y a aquellos que prefieren no planificar su viaje y descubrir el destino una vez allí.

- La aplicación *Inventrip* es multilingüe y detecta automáticamente el idioma del teléfono del turista. Actualmente se encuentra traducida a múltiples idiomas, incluyendo el ruso y el chino.

- Además, *Inventrip* permite conectarse a las señales inteligentes sin necesidad de tener conexión a Internet, aspecto esencial para su despliegue en las zonas rurales de la Ruta Ribera del Duero y para evitar los costes de *roaming* de los cada vez más numerosos turistas extranjeros que visitan la ruta.

- El gestor del destino turístico puede actualizar de manera muy sencilla y en tiempo real la información ofrecida por la señalización conectada desde su PC o tableta, sin necesidad de reprogramar los *beacons* desplegados en las señales turísticas. Se permite así la actualización continua de los contenidos distribuidos por la señalización conectada.

Resultados

Un sistema de señalización turística conectado mejora sustancialmente la acogida de los turistas que visitan un destino, ya que dan información relevante y de calidad a los viajeros que los visitan (dónde ir, qué hacer, dónde comer, dónde dormir) y permiten el acceso a servicios personales como, por ejemplo, sistemas de reserva online.
Sumando a la señalización turística tradicional la tecnología Beacons e Inventrip, el destino turístico puede construir un canal de información con el turista donde cada señal funciona como una oficina de turismo digital. Esto permite al destino turístico comunicarse directamente con el turista sin necesidad de intermediarios.

Este sistema es una solución global con gestión local, es decir, que el turista se conecta siempre de la misma forma a la señalización inteligente sea cual sea el destino. Sin embargo, es el gestor de cada destino quien maneja la información que quiere dar al turista en su territorio. Así se eliminan las fronteras digitales entre destinos turísticos y se unifica la experiencia del usuario.

Cómo funciona

Con Inventrip se puede planificar un viaje en el móvil en tan sólo tres pasos.

En primer lugar, se crea una agenda de viaje que puede ser organizada en carpetas por días, ciudades, restaurantes o los temas principales. Seguidamente, se procede a la navegación y a la selección de los lugares que se pretende visitar. Y, por último, puedes compartir tu viaje con familiares y amigos.

En la ilustración, podemos ver un ejemplo de planificación de un viaje a Bilbao para un fin de semana, en el cual encontramos a la izquierda nuestra ruta de viaje, organizada por días. Un mapa con los lugares seleccionados, y en el cual podríamos seleccionar nuevos puntos de interés y añadirlos a nuestro itinerario. Y a la derecha, una leyenda para identificar de forma fácil los distintos símbolos encontrados en el mapa.

Inventrip se encuentra actualmente en una gran parte de ciudades de España, entre las que se encuentran: Álava, Ávila, Bilbao, Burgos, Cáceres, Cuenca, Gran Canaria, Ibiza, León, Málaga…

Capítulo III

ESTADO DEL ARTE

Estudio de la competencia

Este estudio tiene como objetivo buscar herramientas similares o que posean algunas funcionalidades interesantes relacionadas con la aplicación que vamos a desarrollar. A continuación, vamos a analizar algunas de ellas explicando brevemente para qué sirven y exponiendo los pros y los contras que hemos encontrado.

WhatsApp Whenever Wherever

Así es, hay hoteles que ponen a disposición de sus clientes un número de teléfono con WhatsApp para que puedan dirigir sus dudas. En concreto, vamos a analizar la implantación del servicio bautizado como *WhatsApp Whenever Wherever* lanzado por la conocida cadena de hoteles *Barceló*. [6] [7]

¿En qué consiste?

Al llegar al hotel, los clientes reciben una tarjeta con un número de teléfono al que tienen que dirigir sus mensajes. Este servicio está disponible las 24 horas del día y se atienden todas las solicitudes, ya sean de tipo gastronómico, turístico o, incluso, recomendaciones de sitios populares donde salir a tomar algo.

PROS

- Coste. El coste de la implantación de este servicio es prácticamente nulo, ya que únicamente se necesita una tarjeta SIM y un Smartphone para poder activarlo.

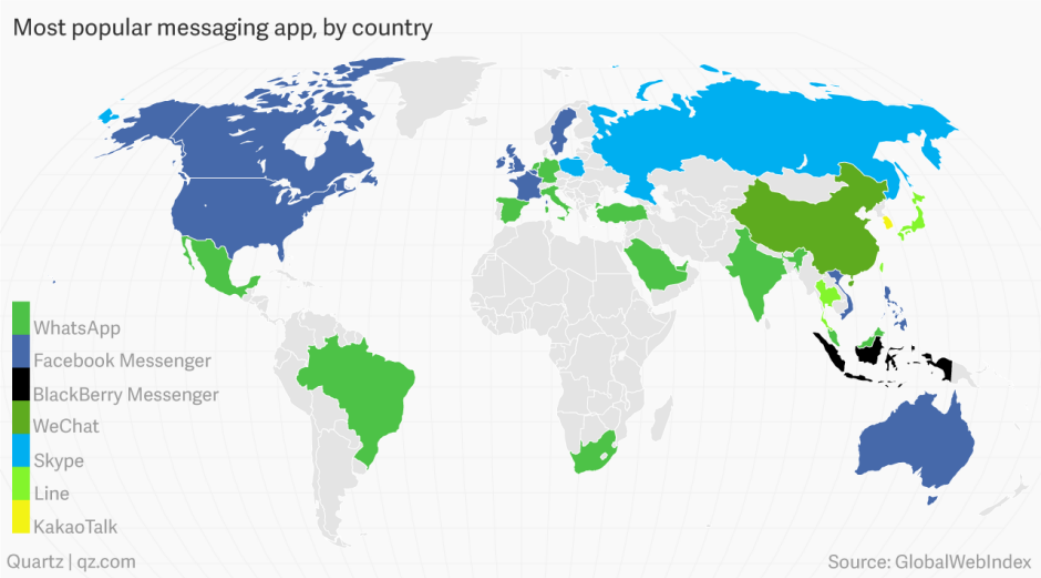
- Usabilidad. Desde 2015, WhatsApp dispone de una versión web que facilita enormemente el mantenimiento de esta plataforma por parte del personal del hotel.

- Disponibilidad. El servicio está disponible las 24 horas del día, lo cual es un punto muy a favor cuando estamos hablando de un hotel.

CONTRAS

- Popularidad entre extranjeros. Aunque la aplicación de mensajería *WhatsApp* es la más utilizada en España, en los hoteles se alojan una gran cantidad de personas procedentes de otros países. Y lo cierto es que, en muchos otros lugares no es la herramienta de comunicación más utilizada.

  A continuación, podemos ver un mapa de un estudio realizado en 33 países y en el que participaron más de 48.000 personas:



  Como se puede ver, en otros muchos países prefieren utilizar otras alternativas. Por otro lado, analizando los datos de turismo registrados en España durante el año pasado, se concluye que británicos, franceses y alemanes fueron los que más visitaron España. Juntando los resultados, sólo en uno de estos países prefieren utilizar *WhatsApp*. [8] [9]

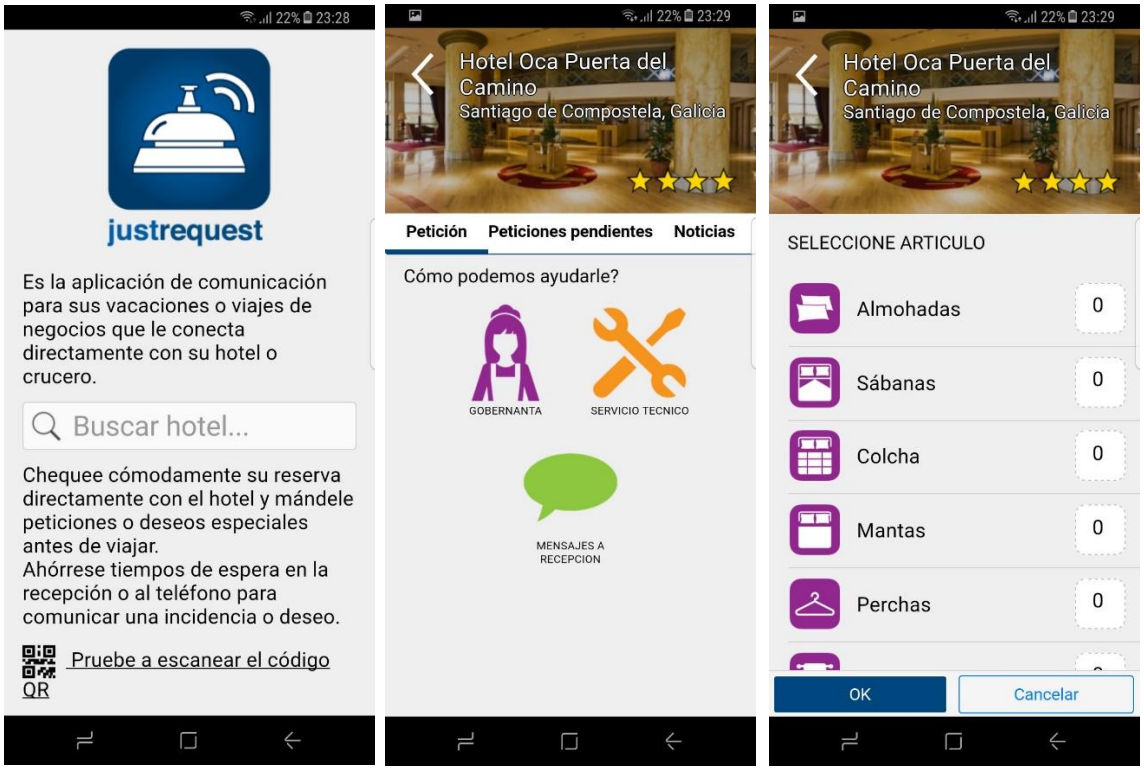
- Profesionalidad. Un sistema de chat propiamente diseñado para hoteles confiere una imagen de mayor seriedad y profesionalidad que *WhatsApp*, permitiendo además añadir funcionalidades extra que requiera el hotel.

- Caídas del servicio. Aunque no se han producido muchas caídas de WhatsApp en el último año, lo cierto es que el hotel está expuesto a ello. La última caída del servicio se produjo el 3 de mayo de 2017 y dejó a los usuarios sin comunicación durante varias horas.

- Disponibilidad de un *Smartphone* con batería y conexión a Internet. Para poder utilizar el servicio de *WhatsApp Web*, los empleados del hotel deberán tener a mano un móvil con batería con el número de teléfono registrado. Ya que, si se quedan sin batería o el teléfono se desconecta de Internet, el servicio de *WhatsApp Web* dejará de funcionar.

Justrequest

Esta idea surgió en 2013 motivada por un hecho que vivió su creador. Éste se encontraba trabajando en su hotel con la recepción llena, cuando unos clientes le dejaron una nota en un papel que decía que necesitaban almohadas para su habitación. Ahí fue cuando pensó que, si los clientes pudieran hacer esto desde su *Smartphone*, se ahorrarían el tener que esperar colas. [10]



¿En qué consiste?

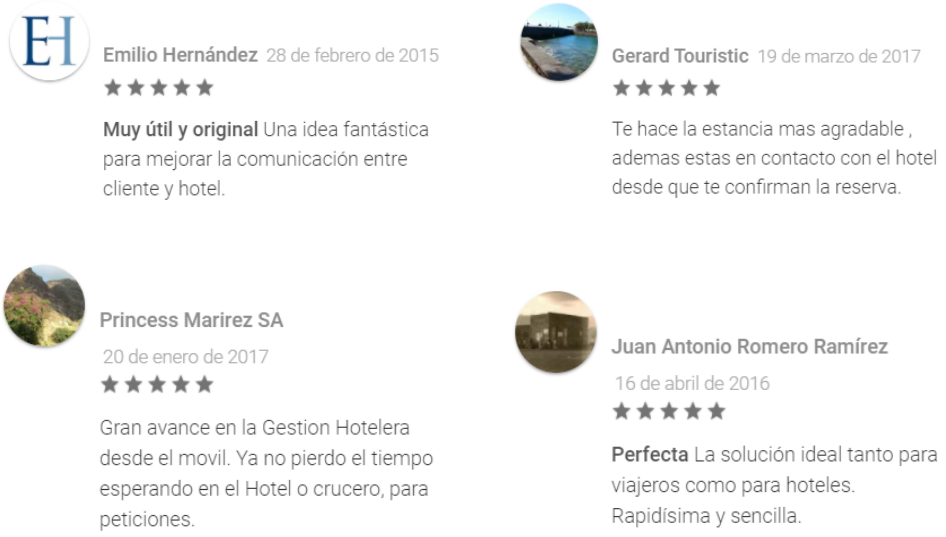
Los clientes pueden acceder a esta herramienta de comunicación, antes y durante su estancia, para solicitar los servicios que necesite sin necesidad de esperar colas en la recepción del establecimiento.

PROS

- Más de 40 funcionalidades. El cliente puede pedir desde un simple paquete de *kleenex* hasta una cama extra cómodamente desde la aplicación.

- Comunicación antes de llegar al hotel. La aplicación permite comprobar el estado de tu reserva y poder pedir servicios incluso antes de llegar al hotel. Por ejemplo, puedes solicitar encontrarte una botella de cava al llegar a tu habitación.

- Valoraciones de los usuarios. Esta aplicación cuenta con la valoración máxima de 5 estrellas en *Google Play Store*. Analizando los comentarios, vemos que la gente está bastante satisfecha con el servicio. Entre los comentarios destacamos el de Emilio que decía que le parecía una idea fantástica para mejorar la comunicación con el hotel o el de Gerard que comentaba que esta aplicación le había hecho la estancia más agradable.



CONTRAS

- Diseño. Desde su creación en 2013, la aplicación no ha actualizado su interfaz por lo que a estas alturas presenta un diseño un poco anticuado.

- No muy extendida en España. El desarrollo de la idea se llevó a cabo en Alemania y aunque la herramienta ya está disponible en España, aún no está extendida por muchos hoteles.

- Control de errores. Si por error se pulsa sobre un artículo, no hay forma de eliminarlo que no sea volviendo a la pantalla anterior. Por ejemplo, si se desea pedir una “toalla de piscina” y se le da sin querer a “toalla de baño”, se tendrá que volver a la pantalla anterior y luego volver a entrar.

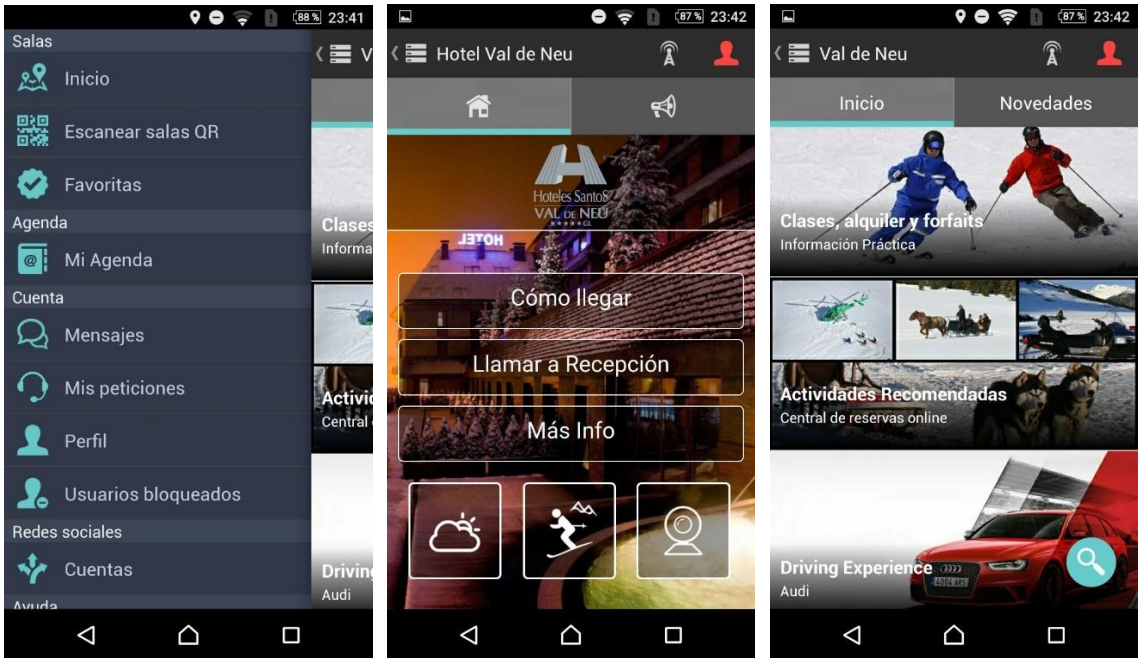
- Usabilidad. La ventaja que se ha comentado antes de que posee más de 40 funcionalidades, puede ser un inconveniente para algunos usuarios ya que, si por ejemplo no funcionan los enchufes de la habitación, se tiene que buscar en el apartado de servicio técnico la acción “chequear enchufes habitación” entre la lista de opciones disponibles.

Virtual Hotel

Virtual Hotel es una herramienta que conecta con los servicios que ofrece el hotel en el que te encuentras hospedado. [11]

¿Cómo funciona?

Con total comodidad y desde el propio Smartphone, se pueden solicitar amenities (jabón, toallas o pasta de dientes), reservar en el restaurante, solicitar un servicio del SPA o pedir que te suban algo de comer a la habitación.



PROS

- **Códigos QR**. Presenta una tecnología muy interesante que se quería utilizar en la implantación del chat de *Inventrip*. Estos códigos QR fueron presentados por el CTO de *Sismotur* como una alternativa a los *beacons* para poder acceder a los chats privados entre la habitación y el hotel.

- **Lista de actividades**. Se le ofrece al cliente una lista de actividades cercanas que se pueden realizar, con imágenes y descripciones detalladas. Lo cual es una funcionalidad muy interesante cuando se habla de hoteles en los que el cliente desconoce la zona en la que se hospeda y quiere echar un vistazo a las actividades de la zona.

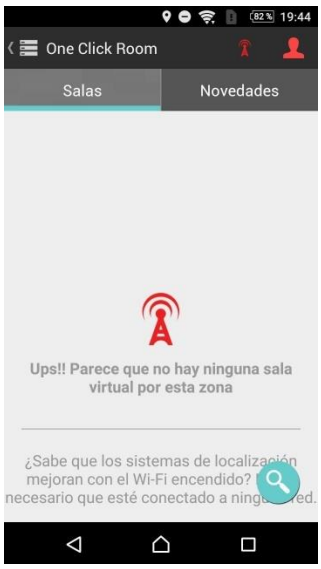
- **Información del hotel**. A parte de las actividades, también se puede encontrar una página de información relacionada con los servicios que presta el hotel.

- **Peticiones pendientes**. El huésped dispone de un apartado de peticiones en el que puede ver en todo momento los servicios solicitados.

- **Bloqueo de usuarios**. El personal del hotel dispone de la funcionalidad de bloquear usuarios si el cliente del hotel utiliza la herramienta de forma maliciosa.

CONTRAS

- **Localización**. El huésped se debe encontrarse físicamente en el hotel para poder ver las salas de chat disponibles, de lo contrario, se le mostrará un mensaje de error diciendo que no se han encontrado salas de chat cercanas. *Justrequest* sin embargo, permitía a los clientes contactar con el hotel antes de llegar a él o cuando salía al exterior durante su estancia.



Mensaje visualizado en la prueba de la aplicación.

- **Compatibilidad**. Se ha probado la aplicación en teléfonos con la versión más reciente de *Android Nougat* y la aplicación se cerraba inesperadamente. Sin embargo, en teléfonos con versiones más antiguas funcionaba sin ningún problema. Su última actualización en *Google Play Store* fue el año pasado y sería necesario que se actualizase para dar soporte a más clientes.

- **Diseño**. Al igual que ocurría con *Justrequest*, la aplicación presenta un diseño anticuado que no se adecúa a la guía de diseño de *Material Design* para el desarrollo de aplicaciones *Android*.

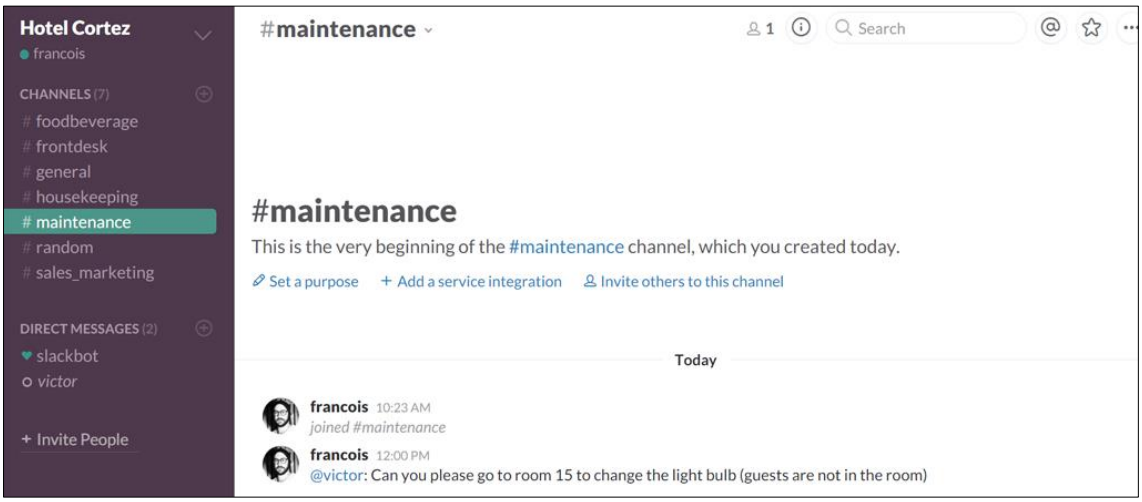
- **Ausencia de versión para ordenador**. No dispone de versión web o de versión de escritorio para facilitar el manejo del chat por los empleados del hotel. Sin embargo, ya hemos visto que otra solución como *WhatsApp,* o *Slack* que veremos a continuación, que sí ofrecen esta utilidad.

Slack

Slack es una herramienta de comunicación popular utilizada por empresas en todo el mundo. Es una alternativa a WhatsApp mucho más completa y profesional. [12]

¿Cómo funciona?

Existen hoteles, como por ejemplo la cadena hotels.ng, que utilizan *Slack* en sus establecimientos adaptándolo a sus necesidades.



PROS

- Aplicaciones nativas para Android, iOS, Mac y Windows. Disponen de aplicaciones en las plataformas más usadas y, a diferencia de WhatsApp, en las versiones de ordenador no se dependerá de la disposición de un teléfono móvil con batería.

- Dos tipos de chats. Se dispone de canales y de mensajes directos. Los canales serán útiles para informar y lanzar ofertas a los clientes del hotel.

- Mensajes programados. Se pueden enviar mensajes automáticos a los usuarios. Esta funcionalidad es muy útil para por ejemplo dar la bienvenida a los nuevos clientes del hotel.

CONTRAS

- Coste. Se presentaba *Slack* como una alternativa a *WhatsApp*. Sin embargo, el servicio de *WhatsApp* es completamente gratuito mientras que en *Slack*, por el volumen de usuarios y las útiles funcionalidades extra, los hoteles se verán en la necesidad de adquirir un plan de pago.

Tecnologías utilizadas

Introducción

Para completar el *Estado del arte*, se describirán las tecnologías utilizadas en el proyecto. Dado que se trata de un proyecto que se va a poner en producción, el coste de estas tecnologías es un factor importante a la hora de ser seleccionadas. Se dedicará un capítulo entero a la evaluación de las dos APIs que se han utilizado en el proyecto.

HTML5

HTML (Hyper Text Markup Language) es un lenguaje de marcado que permite establecer la estructura y contenido de una página web mediante el uso de etiquetas, además de ofrecer la posibilidad de integrar otros elementos como pueden ser imágenes, tablas, listas, enlaces, anclas…

En el chat web, se ha usado una estructura por clases y clases derivadas, que se van agregando/quitando dinámicamente según convenga.

CSS3

Cascading Style Sheet 3 Es un lenguaje de estilos en cascada que permite dar apariencia a la página web, fijando aspectos como el tamaño, el color, la fuente, la disposición y otros comportamientos dinámicos como mostrar ocultar información cuando el ratón pasa por encima de un elemento.

Se ha prestado atención a la estructura de clases en HTML para conseguir una consistencia absoluta durante todo el proyecto y además así ahorrar líneas de código CSS.

JavaScript y jQuery

JavaScript es el lenguaje de programación predominante en el chat. Controla dinámicamente todas las interacciones del usuario con la vista, enviando datos a las bases de datos y devolviendo resultados.

Se ha puesto especial atención en conseguir un modelo consistente y parametrizado, con el fin de generar un código lo más limpio posible y legible. La gestión de eventos se realiza siempre de forma dinámica, agregando o quitándolos según convenga.

También se ha usado la biblioteca jQuery, una simplificación de JavaScript muy cómoda a la hora de obtener selectores HTML para ser tratados de la forma deseada, ya sea para agregar o quitar contenido como para ocultar o mostrar elementos.

SendBird

SendBird se define como una herramienta de mensajería potente y totalmente configurable. [3]

Esta herramienta posee las principales características necesarias para realizar nuestro chat e integrarlo en la app de *Sismotur*. Además, *SendBird* tiene una amplia documentación.

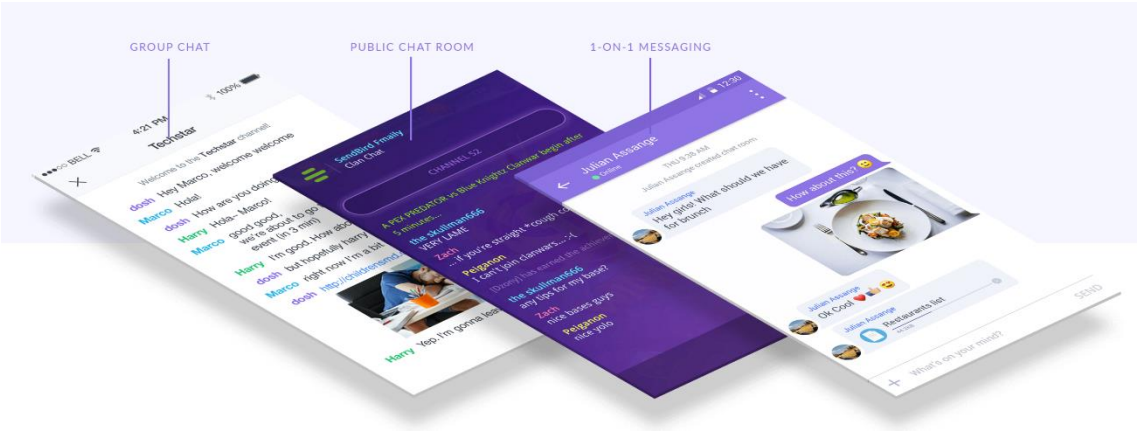
Todos los ejemplos que se van a utilizar, partirán de la base que la aplicación es usada por clientes de un hotel, el cual se comunica a través de la recepción.

La herramienta dispone de los siguientes tipos de chat:

- 1 -on-1 messaging. Conversaciones corrientes entre 2 personas. En nuestro caso, podría ser una conversación entre el cliente de la habitación 37 y el recepcionista del Hotel.

- Group chat. Conversaciones entre varios usuarios. En nuestro caso, la recepción del hotel con un grupo de habitaciones, un par de parejas que han ido juntos de vacaciones y les interesaría recibir la información de forma conjunta.

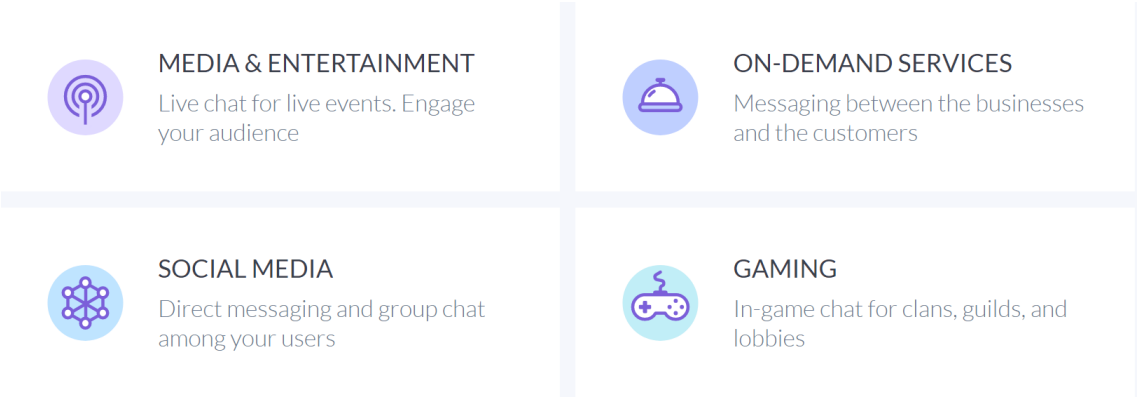
- Public chat room. Salas de chat abiertas a cualquier usuario de la aplicación. En nuestro caso podría ser una sala con información del hotel, o con información sobre los próximos eventos de la ciudad, para que los huéspedes estén enterados de las posibles actividades que pueden realizar.



Viendo estos 3 tipos de chats, *SendBird* demuestra la solidez de su servicio, ya que permite tanto conversaciones de usuarios limitados, como conversaciones masivas entre usuarios como podría ser el canal público de Madrid, en el cual podrían hablar una gran cantidad de usuarios de forma simultánea.

Diferentes sectores que utilizan *SendBird*

SendBird es utilizado en sectores como el entretenimiento, las redes sociales, juegos en línea y la demanda de servicios. En este último sector, se situaría *Sismotur*, ya que su idea es ofrecer un servicio con el cual facilitar y mejorar la experiencia del turista durante su viaje.



Como podemos ver, *SendBird* posee en un amplio abanico de posibilidades, por lo que se adapta a la mayoría de aplicaciones que necesiten de un proveedor de chat.

En nuestro caso, las principales utilidades que *SendBird* nos podría proporcionar serian:

- Bot Interface: *SendBird* nos da la posibilidad de diseñar y personalizar nuestro propio bot, con el cual nuestro hotel podría responder a dudas generales de manera sencilla, o de informar del horario de apertura del restaurante.

- Push notificactions: Nos proporciona un servicio de notificaciones, con el cual el usuario no tendria que acceder a la aplicación para leer los mensajes.

- Admin messages: Permite enviar mensajes como si fueran alertas, de esta manera se intenta que el mensaje sea leido lo antes posible por el usuario. Para mostrar avisos importantes del hotel, como retraso a la hora del check-out.

- Spam flood protection: *SendBird* nos permite prevenir el spam en salas de chat, de manera que un usuario no estropee la armonía de la sala con múltiples mensajes o con mensajes de publicidad

- Auto translation: La última función que vamos a comentar, es probablemente la más importante para nuestro proyecto, y es la que nos permite traducir de forma instantánea mensajes entre usuarios. Gracias a esta función, la gente podría

intercambiar mensajes por las salas de chat de forma fácil y eficiente, así como los empleados del hotel solventar las dudas de sus clientes.

Estas son solo algunas de las características que ofrece *SendBird*, viendo el potencial que tenía, decidimos comenzar a realizar el proyecto con este servicio. Tras estudiar la documentación y hacer un programa de prueba, contactamos con los del apartado de ventas de *SendBird*, ya que de forma gratuita solo podíamos utilizar parte de los servicios, con un número limitado de usuarios.

Start with the FREE plan. Consult our sales team for a custom pricing model that works for your business.	FREE GET STARTED	CUSTOM PRICING CONTACT US
CORE FEATURES		
1-on-1 Messaging, Group Chat, Open Channels	✓	✓
PREMIUM FEATURES		
Message Retrieval API	✗	✓
Data Export Feature & API	✗	✓
Moderation Tools	✗	✓
Auto-Translation	✗	✓
Auto-Thumbnail Generation	✗	✓
Bot Interface	✗	✓
Migration API & Support	✗	✓
Spam Flood Protection	✗	✓
File Encryption	✗	✓
VOLUME QUOTA		
Message Retention	3 Months	Custom
MAU Limit	1,000	Custom
Peak Connections Limit	25	Custom
Storage / Traffic	Up to 5GB / month	Custom
File Size Limit	Up to 5MB / file	Custom
SUPPORT PROGRAMS		
Response Times	Best Efforts	Priority (Custom)
Service-Level Agreements	✗	✓
Priority Feature Support	✗	Custom
Integration Support	✗	Custom
Billing	–	Credit Card or Invoice

Funcionalidades de pago

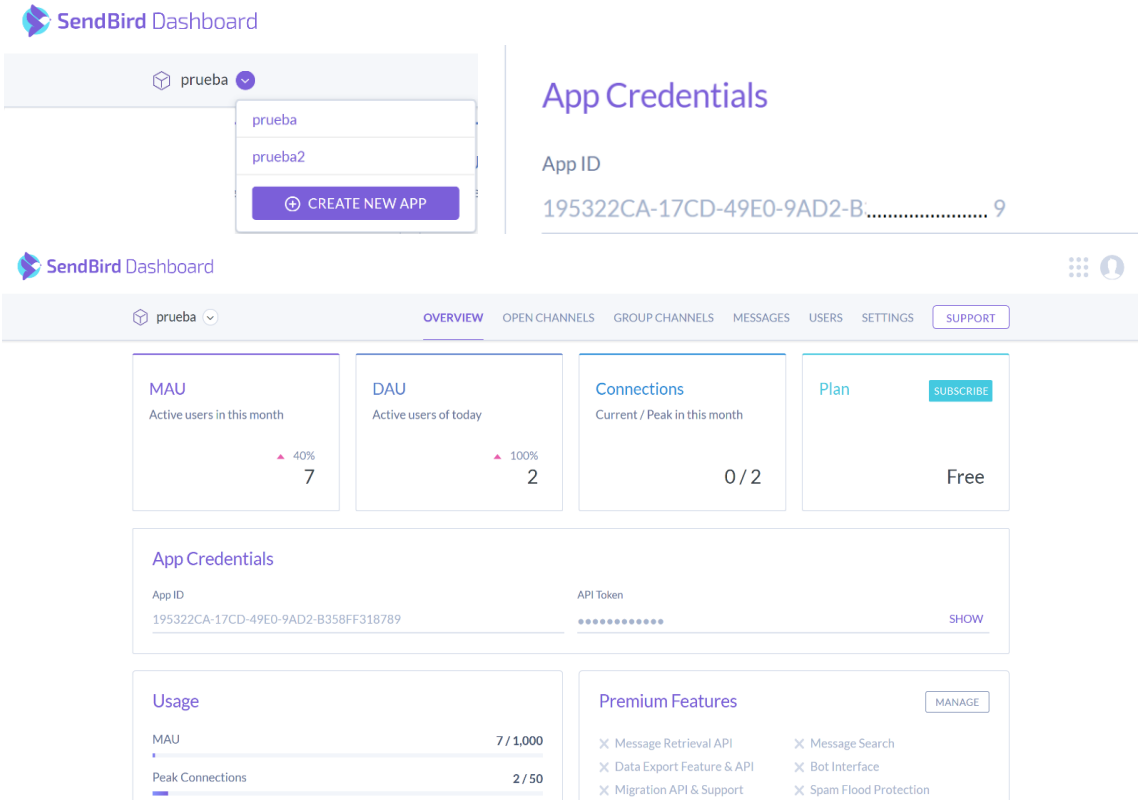
La mayoría de las funcionalidades que ofrece SendBird las incluye en el plan de pago, por lo que nuestras soluciones se limitaban a implementar un chat simple con usuarios muy limitados.

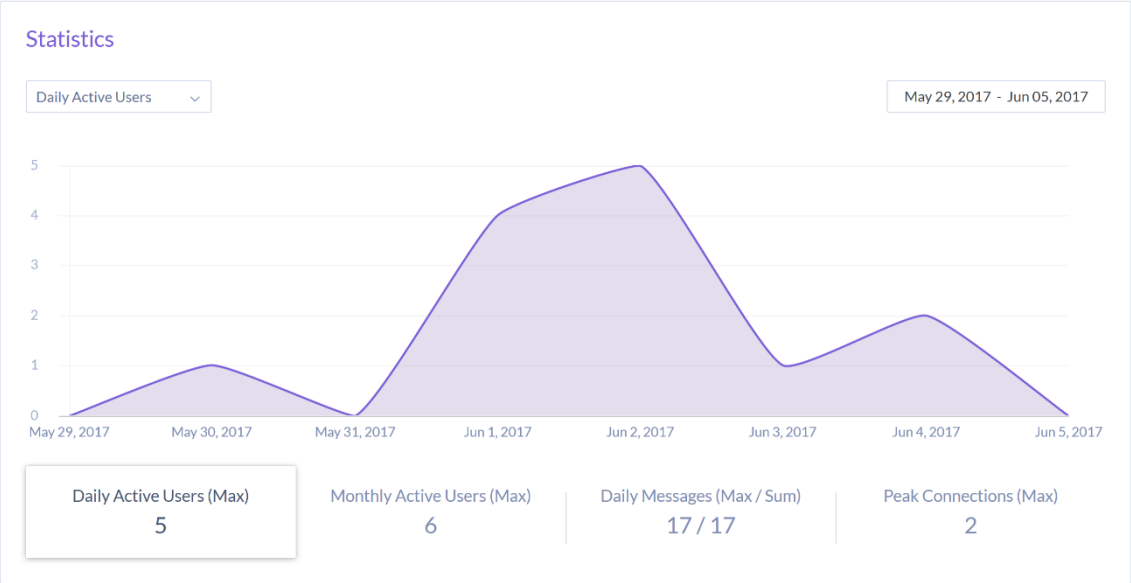
Contactamos con el departamento de ventas de SendBird para ver qué precio tendría la contratación de los servicios de pago, los cuales, según nos comentaron empezaban alrededor de los 600 euros mensuales.

Cómo se integra en un proyecto

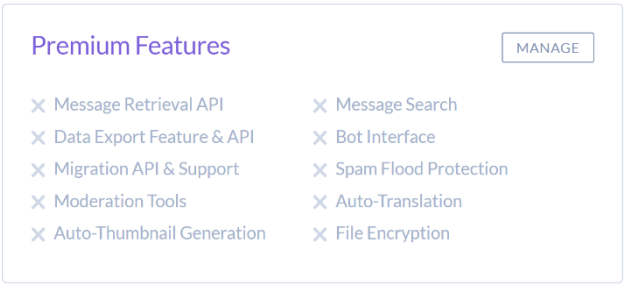
La forma de integrar *SendBird* en un proyecto a priori es bastante sencilla, ya que nos proporciona un caso de ejemplo, tanto para web, como para Android e iOs.
De manera que puedes comenzar a utilizar su versión de prueba casi al instante.

Esta versión, dispone de una serie de chats y salas abiertas de modo que puedas ver el funcionamiento básico de la aplicación, e incluso chatear con otra gente que este probando el programa, ya que cada aplicación posee un *App ID*.
Cuando creas una App se te otorga un App Id propio, así como una base de datos con usuarios totalmente funcional, y el acceso a tu propio *Dashboard* para controlar el uso de tu aplicación.





Después de realizar las modificaciones del App Id en tus proyectos, para que parte web y móvil se conecten y se pueda chatear es necesario que tengan el mismo *App ID*, y probar que la aplicación funciona, se puede comenzar a añadir modificaciones y funcionalidades a tu programa.



SendBird te muestra en todo momento las mejoras disponibles y cuáles de ellas tienes contratadas.

En el caso de este proyecto, al ser la versión más básica, no se ha contratado ningún servicio.

Para contratar algún servicio *premium*, se debe contactar con el departamento de ventas, los cuales te proporciona las mejoras y te guían durante el proceso de integración de estas funciones, o así nos lo han dejado entender en los correos que hemos intercambiado con ellos.

Chat SDK

Chat SDK es un SDK de mensajería instantánea para iPhone y Android. Es de código abierto gratuito y fácil de integrar en cualquier aplicación, del que tendremos el control total sobre los datos de la aplicación y el código. [4]

El funcionamiento de este SDK se basa en *Firebase* (descrita más adelante), lo que beneficia para poder comunicarnos con la versión web necesaria en nuestro proyecto.

Funcionalidades:

- Chats Privados: conversaciones privadas entre dos personas convertibles en chats de Grupo simplemente invitando a otra persona.

- Chats de Grupo: conversaciones entre un número de personas ilimitadas.

- Chats Abiertos: conversaciones abiertas a todos los usuarios de la aplicación, contienen información general.

- Login Flexible: permite registrarse con correo electrónico, Facebook, Twitter o de manera anónima por lo que no se necesita conectar siempre desde el mismo dispositivo.

- Diferentes tipos de mensajes: Chats SDK permite enviar diferentes tipos de mensajes como texto, imagen, video, audio y localización. Algunos de ellos hay que pagarlos e integrarlos aparte.

- Acceso total a los datos: al estar implementado sobre Firebase se tiene un control total sobre los datos simplemente entrando en la cuenta de firebase, donde además podremos analizar la utilización de nuestra aplicación.

- Código fuente completo: esta es una de las características por la que se ha decidido utilizar Chat SDK en vez de SendBird, ya que se tiene acceso total al código, está bien comentado, documentado y es fácil de modificar.

Precio

Una de las razones por la que se ha decidido utilizar *Chat SDK* es el precio, ya que la mayoría de las funcionalidades que se necesitan para nuestra aplicación son gratuitas y el único gasto sería el alojamiento en *Firebase*, el cual para un número menor a 100.000 usuarios es gratuito.

Esta sería la tabla de precios de *Firebase*:

tamaño aplicación	NIVEL FIREBASE
Pequeña aplicación - bajo 1k usuarios diarios	GRATIS
tamaño medio de aplicación - en virtud de 10k usuarios diarios	GRATIS
popular aplicación - 100k usuarios diarios	$ 199 POR MES
Mayor aplicación - más de 100 mil usuarios diarios	$ 199 POR 100 MIL USUARIOS AL MES

Aparte de las funcionalidades gratuitas, *Chat SDK* ofrece otras funcionalidades de pago de las cuales algunas podrían ser útiles para la aplicación como:

- Chats basados en la ubicación: lo que ofrece es una lista de usuarios cercanos, algo que sería muy recomendable para nuestra aplicación en casos como turistas que estén visitando el mismo lugar o que estén en el mismo hotel.

- Mensajes de video: muy útil para recomendar a usuarios cercanos sitios o lugares que visitar y que ellos puedan decidir por sí mismos si visitarlos o no.

- Confirmaciones de lectura: para ver cuando un mensaje ha sido leído, muy cuando hablamos a un hotel y poder ver si han visto ya nuestra petición.

Firebase

Firebase es una tecnología gratuita que permite al usuario mejorar sus aplicaciones de forma escalable, ya sean en formato web o móvil. Al utilizar la infraestructura de *Google*, proporciona una manera sencilla de monitorizar los datos de la aplicación, sin tener que preocuparte por la demanda de usuarios. [5]

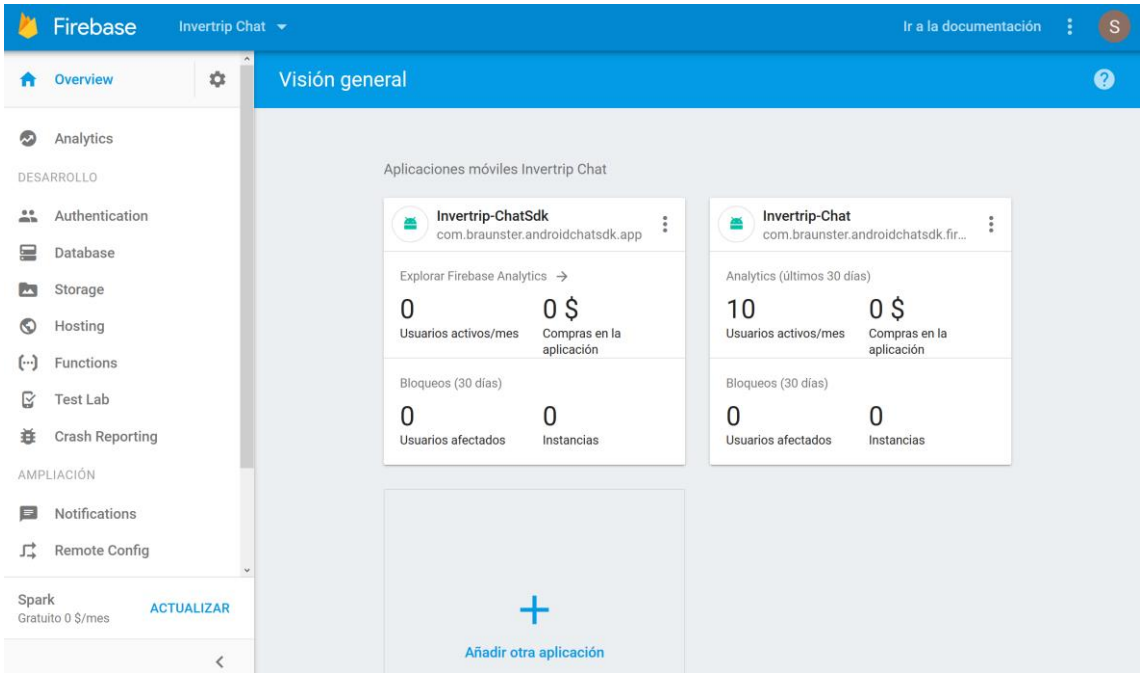
Para monitorizar los proyectos, esta SDK viene acompañada de una consola, donde el creador puede agregar y gestionarlos como desee. Esta consola presenta una interfaz muy amigable.

Cómo se integra en un proyecto

Nada más entrar en la página web de *Firebase*, el programador puede comenzar su uso de forma casi inmediata. Basta con crear una cuenta, añadir un proyecto y configurar la aplicación de una manera muy sencilla:

- Si se trata de un proyecto web, basta con añadir un poco de código al inicio de nuestro archivo *JavaScript*.

- En el caso de un proyecto Android o iOS, hay que realizar algunos pasos adicionales, pero todo se encuentra correctamente documentado.



La documentación ofrecida por *Firebase* es muy amplia y está traducida a muchos idiomas. Además, posee gran variedad de ejemplos de uso para cada una de las funcionalidades descritas.

Funcionalidades

Analíticas. Contiene información sobre la rentabilidad de la aplicación, como el promedio de ingresos, los usuarios que tienen la aplicación, los compradores. Es una herramienta muy útil para monitorizar la evolución de la aplicación en términos económicos.

Autenticación. Contiene información sobre todos los usuarios registrados, como el último inicio de sesión, el identificador de usuario, la fecha de registro, etc. Adicionalmente, nos proporciona una herramienta para poder gestionar los métodos de inicio de sesión disponibles (correo electrónico, *Google, Facebook, GitHub, Twitter,* anónimo), así como una forma sencilla de enviar correos electrónicos de confirmación y de cambio de contraseña.

Este sistema es un punto fuerte a favor de *Firebase*, especialmente el inicio de sesión anónimo, que permitiría a un cliente poder registrarse y preguntar en un canal público cualquier cosa sin necesidad de registrarse.

En *JavaScript,* podemos obtener los métodos de autenticación usando Firebase.auth().

Database. *Firebase* proporciona una herramienta de base de datos asíncrona y muy flexible. Funciona por medio de un archivo de extensión JSON con estructura arbórea que podemos modificar de forma manual o dinámica desde nuestra aplicación.

Adicionalmente, contiene un archivo con los permisos que queremos dar a los usuarios, pudiéndolos cambiar en cualquier momento. La sintaxis de este archivo es parecida a *JavaScript*, por lo que de inmediato el creador puede probar su eficacia, mediante el uso de un simulador de consultas. La base de datos también presenta un sistema de monitorización de conexiones, espacio de almacenamiento y descargas.

La forma de acceder a *Firebase Database* es bastante trivial: con el método Firebase.database() disponemos de un número variado de consultas asíncronas (en segundo plano) que devuelven una Promesa: cuando se resuelva el estado de la consulta, procederemos a ejecutar el código que deseemos. En JavaScript, la manera de usar el objeto devuelto por la consulta es mediante la notación de acceso a cualquier objeto en formato JSON, teniendo en cuenta que para obtener la clave de la consulta debemos usar el atributo *key* y para acceder al valor, el método *val().*

Storage. Esta herramienta de almacenamiento de archivos resulta útil cuando se trata de querer guardar imágenes de perfil o simplemente imágenes enviadas como mensaje de texto. *Firebase* permite acceder a *storage* con Firebase.storage().

Adicionalmente, tenemos un archivo de configuración de reglas, donde podremos cambiar los permisos de los usuarios a la hora de manipular *storage*.

Hosting. *Firebase* incluye un servicio de hosting. Podemos solicitar un dominio, verificarlo y publicar nuestro proyecto. También podremos monitorizar datos como el almacenamiento y las descargas.

Otras funcionalidades. Disponemos también de un registro de errores en la consola de *Firebase*, un componente llamado *AdMob* para añadir anuncios a la aplicación, un servicio de configuración remota del servidor para no tener que pedirle al usuario que actualice la aplicación.

Capítulo IV

PROCESO DE DESARROLLO

Metodología de trabajo

Este Trabajo de Fin de Grado ha seguido ciertas pautas conocidas en proyectos de Ingeniería Informática. En este punto se explica en profundidad en qué han consistido y cómo se han llevado a cabo. Cabe recalcar que se ha creado un canal de comunicación entre Felipe Santi (CTO de *Sismotur*), Juan Antonio Recio García (director del proyecto) y los integrantes del grupo.

Metodologías ágiles

A lo largo del proyecto se han puesto en práctica el desarrollo ágil de software, en el que predomina un enfoque incremental e iterativo. A la hora de tomar decisiones a corto plazo, siempre se ha intentado buscar un resultado óptimo en torno a los requisitos, que han ido evolucionando con el tiempo.

Cada una o dos semanas se han tenido reuniones en el despacho del director, en el que se han discutido las tecnologías a utilizar y el progreso relativo a la reunión. En cada reunión se ha ido fijando el día y hora de la siguiente, y en el caso de que alguna de las partes del proyecto no estuviera disponible, siempre se ha trabajado con alternativas.

En particular, en algunos casos los alumnos se han reunido en la sala de estudio de la biblioteca de la Facultad de Informática, o se han hecho videoconferencias en *Skype*.

De esta manera, en cada reunión se han pensado nuevos requisitos, que han sido analizados y posteriormente se han discutido planes de acción. Tras la implementación, se han ido encontrando problemas, más tarde discutidos en las siguientes reuniones hasta dar con las soluciones.

Herramientas de desarrollo

Skype

Esta popular herramienta es indiscutible cuando se trata de comunicarse entre miembros del equipo en el caso de no poder personalmente. En cada reunión periódica, se ha establecido una videoconferencia con Felipe Santi, donde se han discutido y tratado los progresos gracias a su funcionalidad de compartir pantalla.

Slack

Este programa ha resultado de utilidad a la hora de expresar formalmente problemas e ideas para el proyecto. *Slack* se ha usado especialmente en la comunicación con Felipe Santi, quien creó el grupo de trabajo en el programa.

GitHub

Es el sistema de control de versiones más usado por programadores. Al comienzo del proyecto se creó un repositorio privado con los integrantes del proyecto, donde se han ido subiendo actualizaciones de la aplicación.

Para realizar las actualizaciones, se ha optado por instalar *GitHub Desktop*, ya que resulta una manera cómoda y visual de ver y repasar los cambios y nuevas implementaciones.

Diseño de la interacción del sistema

Factor de forma

Este proyecto ha sido pensado para ser escalable, y hoy en día la forma más cómoda de integrar un chat para todos los ciudadanos, es mediante tecnologías móviles. El teléfono móvil es algo que uno lleva consigo a diario a todos los lugares y actividades que realice en su vida cotidiana.

Además, como ya se ha indicado, se necesita una versión con un formato mayor, como puede ser una página web o una *tablet*. Esto se debe a la comodidad de la que dispondría la recepción de los hoteles a la hora de contactar con las habitaciones.

Postura

La postura del usuario variará según el rol.

- Un turista necesitará el uso del chat de manera momentánea. Siempre que le surja una duda, o requiera un servicio, podrá comunicarse con la recepción y minimizar *Inventrip*, por lo que su postura será temporal.
- Un usuario recepcionista deberá estar disponible en todo momento, comprobando si ha habido nuevos mensajes en los chats de las habitaciones o en los canales públicos. Por lo tanto, su postura será soberana.

Métodos de entrada

Inventrip está disponible para *smartphone*, por lo que el sistema de entrada del chat es táctil. Se ha optado por una interfaz minimalista, con botones claramente visibles y siguiendo determinadas directrices de diseño. De esta manera, el usuario podrá interactuar de forma sencilla con la aplicación sin encontrar muchos problemas.

La versión web del chat, presenta una interfaz diferente, más amplia y evidentemente destinada para ser usada en un ordenador. Es por esta razón que la entrada, además de ser táctil, se realizará por medio de ratón y teclado físico.

Requisitos funcionales

A continuación, se expondrá una lista de casos de uso interesantes para el chat.

Chat de habitación – Turista

- Registrarse.
- Iniciar sesión.
- Cerrar sesión.
- Entrar en un chat público.
- Entrar en un chat privado.
- Enviar mensaje (texto, foto u otro archivo).
- Visualizar archivo.

Chat de habitación – Recepcionista

- Registrarse.
- Iniciar sesión.
- Cerrar sesión.
- Enviar mensaje (texto, foto u otro tipo de archivo).
- Enviar mensaje (texto, foto u otro tipo de archivo) difundido a todas las habitaciones.
- Enviar mensaje (texto, foto u otro tipo de archivo) a habitaciones concretas. De la misma manera que el caso anterior, puede existir la posibilidad de tener que enviar un mensaje concreto a las habitaciones suscritas a un servicio determinado.
- Eliminar mensaje propio.
- Eliminar mensajes de todas las habitaciones.
- Visualizar archivo.

- Filtrar habitaciones.
- Visualizar lista de usuarios de una habitación.
- Filtrar usuarios de un chat.
- Visualizar lista de usuarios del hotel.
- Filtrar usuarios del hotel.
- Invitar usuario a chat de habitación.
- Eliminar usuario de chat de habitación.
- Crear chat de habitación
- Crear chat de varias habitaciones
- Renombrar nombre del chat.
- Salir de chat público.
- Eliminar chat de habitación.

Elementos de datos

Acorde con el capítulo III (Requisitos funcionales), en esta sección se procederá a generar una lista de elementos de datos de la interfaz. La finalidad de este apartado es proporcionar una guía a la hora de implementar todas las funcionalidades del chat.

Android

Elemento *canal público*

Presenta los siguientes atributos:

- Nombre
- Imagen

Está relacionado con la lista de canales grupales

Elemento *lista de canales públicos*

Presenta los siguientes atributos:

- Canales públicos
- Añadir canal público

Elemento *usuario*

Presenta los siguientes atributos:

- Nombre
- Imagen de perfil

Está relacionado con la lista de usuarios

Elemento *lista de usuarios*

Presenta los siguientes atributos:

- Buscador
- Usuarios
- Seleccionar usuario
- Agregar usuarios

Elemento *canal grupal*

Presenta los siguientes atributos:

- Nombre
- Imagen

Está relacionado con la lista de canales grupales.

Elemento *lista de canales grupales*

Presenta los siguientes atributos:

- Canales grupales
- Añadir canal grupal

Elemento *perfil*

Presenta los siguientes atributos:

- Imagen de perfil
- Nombre de usuario
- Teléfono
- Correo electrónico
- Cerrar sesión

Elemento *cabecera*

Presenta los siguientes atributos:

- Nombre del chat
- Invitar usuarios
- Lista de usuarios

Está relacionado con el chat.

Elemento *mensaje*

Presenta los siguientes atributos:

- Nombre de usuario
- Fecha y hora

- Mensaje

Está relacionado con la lista de mensajes.

Elemento *lista de mensajes*

Presenta los siguientes atributos:
- Mensajes

Está relacionado con el chat.

Elemento *enviar mensaje*

Presenta los siguientes atributos:
- Texto del mensaje
- Subir archivo

Está relacionado con el chat.

Elemento *chat*

Presenta los siguientes atributos:
- Cabecera
- Lista de mensajes
- Enviar mensaje

Elemento *diálogo modal*

Presenta los siguientes atributos:
- Título
- Descripción
- Campos
- Cancelar
- Aceptar

Web

Elemento *canal público*

Presenta los siguientes atributos:
- Nombre
- Imagen

Está relacionado con la lista de canales grupales

Elemento *lista de canales públicos*

Presenta los siguientes atributos:

- Descripción
- Canales públicos
- Añadir canal público

Elemento *usuario*

Presenta los siguientes atributos:

- Nombre
- Imagen de perfil
- Seleccionado

Está relacionado con la lista de usuarios

Elemento *lista de usuarios*

Presenta los siguientes atributos:

- Buscador
- Usuarios

Elemento *canal grupal*

Presenta los siguientes atributos:

- Nombre
- Notificación

Está relacionado con la lista de canales grupales.

Elemento *lista de canales grupales*

Presenta los siguientes atributos:

- Canales grupales

Elemento *perfil*

Presenta los siguientes atributos:

- Nombre de usuario
- Imagen de perfil
- Editar perfil

Elemento *cabecera*

Presenta los siguientes atributos:

- Nombre del chat
- Invitar usuarios
- Lista de usuarios

- Editar chat
- Salir de chat
- Eliminar chat

Está relacionado con el chat.

Elemento *mensaje*

Presenta los siguientes atributos:
- Nombre de usuario
- Fecha y hora
- Mensaje
- Eliminar mensaje

Está relacionado con la lista de mensajes.

Elemento *lista de mensajes*

Presenta los siguientes atributos:
- Mensajes

Está relacionado con el chat.

Elemento *enviar mensaje*

Presenta los siguientes atributos:
- Texto del mensaje
- Subir archivo

Está relacionado con el chat.

Elemento *chat*

Presenta los siguientes atributos:
- Cabecera
- Lista de mensajes
- Enviar mensaje

Elemento *diálogo modal*

Presenta los siguientes atributos:
- Título
- Descripción
- Campos
- Cancelar
- Aceptar

Diseño

Paleta de colores

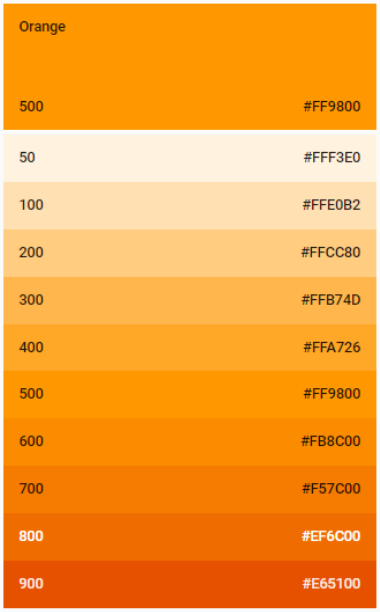
Para llevar a cabo la elección de colores del sistema de chat, se ha seguido la guía de *Material Design* de *Google*. Este apartado de colores, según *Google*, es aplicable tanto a *Android* como a Web e *iOS*.

Tanto en la página web de *Inventrip* como en la parte de la aplicación que ya tienen desarrollada, el color predominante es el naranja. Se ha visto que no utilizan una tonalidad fija de naranja, si no que tanto para el logo, como para la aplicación o como en su página web, utilizan diferentes tonos de naranja sin seguir ningún patrón.

Por esta razón, para escoger los colores de nuestro sistema de chat se ha decidido seguir las directrices que marca *Google* en su guía de diseño.

Para ello, los pasos que se han seguido, han sido los siguientes:

1. *Google* sugiere el uso de los colores 500 como los colores primarios en la aplicación. Así, se ha escogido el color 500 (#FF9800) como color predominante.
2. El resto de colores de la paleta pueden ser combinados con el anterior. Siguiendo esta línea, se ha escogido un tono dos niveles por debajo, el color 700 (#F57C00). Este color se puede ver, por ejemplo, contrastando la división entre la barra de estado y la barra de herramientas del diseño móvil de la segunda iteración.



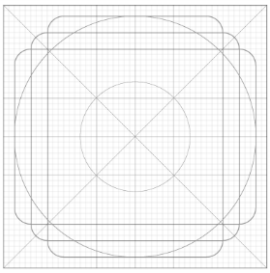
3. También se puede utilizar un color secundario para acentuar partes clave de la interfaz de usuario. En este caso se ha escogido usar una tonalidad de azul utilizada por *Inventrip*.

Icono

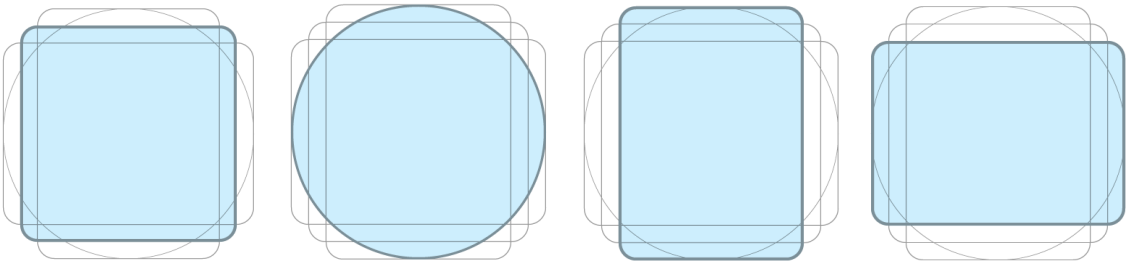
De la misma manera que con los colores, se ha decidido seguir la guía de *Material Design* de *Google* para diseñar el icono de nuestra aplicación móvil.

Los iconos materiales utilizan formas geométricas para representar las ideas básicas, capacidades o temas. Por lo que, al buscar el CTO de *Sismotur* un icono simple que diera a entender a la gente la idea de un sistema de comunicación, se optó por esta vía.

Para ello se partió de la siguiente plantilla de icono estándar que se proporciona en la guía de estilos:



En esta plantilla se ofrecen varias opciones al desarrollador para que pueda crear su icono de forma circular, rectangular o cuadrada. Nosotros optamos por esta última forma ya que es la más común en las aplicaciones hoy en día.



Cuadrado *152dp x 152dp* ***Círculo*** *176dp* ***Rectángulo vert.*** *176dp x 128dp* ***Rectángulo hor.*** *128dp x128dp*

Con la base cuadrada de 152dp x 152dp, utilizamos el denominado color 500 de la paleta naranja para el fondo, posteriormente dibujamos en el interior un "bocadillo" de mensajería para dar a entender la idea de un sistema de chat junto con la abreviatura de *Inventrip* en el centro. De este modo, se consiguió el diseño minimalista que se buscaba.

El resultado final fue el siguiente:

Implementación con Sendbird

Interfaz móvil

La versión móvil presenta un diseño sencillo y funcional, con el fin de que el cliente que llega al hotel pueda hacerse rápidamente con la aplicación y poder chatear con el hotel para satisfacer sus necesidades.

Por ello, se optó por añadir dos pestañas que contienen los diferentes chats del hotel:

- La primera **pestaña Grupos** contiene los chats *"1 a 1"* (entre el hotel y el cliente de una habitación) y los chats *"n a n"* (dados en el caso excepcional de que se alojen un grupo de conocidos en diferentes habitaciones del hotel).

- La segunda **pestaña Canales** contiene los chats dedicados a difundir mensajes a un importante número de usuarios, muy útiles para la difusión de promociones.

Imagen: Vista de lista de grupos

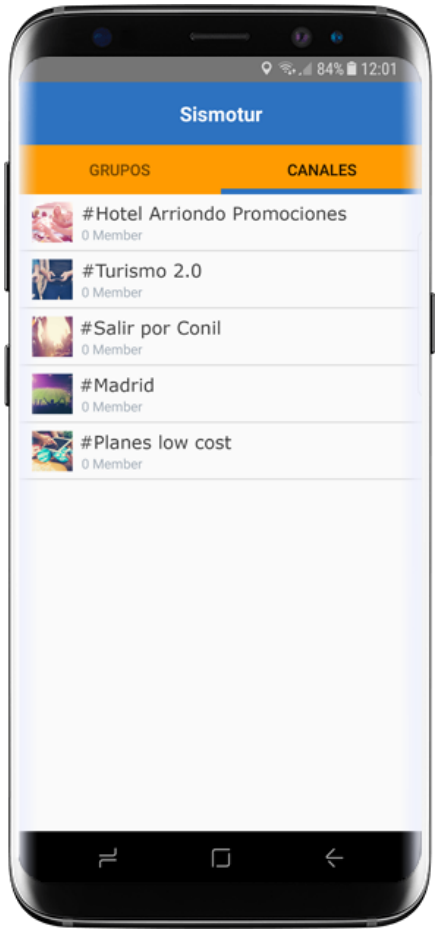


Imagen: Vista de lista de canales

Si seleccionamos un grupo, accederemos a la **vista del chat**. Esta vista presenta un diseño muy similar a *WhatsApp* y a las aplicaciones de mensajería actuales con las que los clientes del hotel estarán acostumbrados a comunicarse.

En ella se pueden ver los mensajes de los usuarios junto con la imagen de perfil, la fecha en la que fue enviado el mensaje y el nombre asignado a la otra persona que habla. También se ha optado por diferenciar el color de los bocadillos entre el emisor y el receptor utilizando el característico color naranja de *Inventrip*.

Imagen: Vista de una conversación

De forma análoga, si seleccionamos un canal podremos ver una vista de chat muy similar que utiliza el mismo diseño.

Otras de las funcionalidades que se añadieron al sistema de chat, fueron la posibilidad de recibir y visualizar en el móvil, archivos en formato PDF, imágenes y vídeos. Esto será útil de cara a que el hotel pueda enviar información y promociones a sus clientes.

En primer lugar, se comenzó implementando la funcionalidad para visualizar archivos PDF. Al pensar cómo implementarlo, se observó que se podía dar el caso de que el cliente no tuviese instalado en un Smartphone una aplicación para visualizar dichos archivos.

Por ello, se distinguieron los siguientes casos y se decidió aplicar la solución que mejor satisficiera al cliente:

- Si el cliente dispone de un visor de archivos PDF instalado, la aplicación lo abrirá directamente con dicha aplicación cuando finalice la descarga. Al pulsar sobre la miniatura, se abrirá un cuadro de diálogo que el usuario deber aceptar para comenzar la descarga.

Imagen: PDF recibido

Imagen: Confirmación de descarga

Durante la descarga, se muestra una ventana para dar a conocer al usuario que el archivo se está descargando. Deslizando la barra de notificaciones hacia abajo, se podrá ver una barra de progreso de la descarga en curso.

Imagen: Feedback de espera

Imagen: Elección de apertura de app

Si el usuario dispone de varias aplicaciones para visualizar archivos PDF en su dispositivo, se le mostrará una ventana para que elija con qué aplicación desea abrirlo en el caso de no tener fijada una aplicación como predeterminada.

Finalmente, el cliente podrá visualizar el archivo en su teléfono:

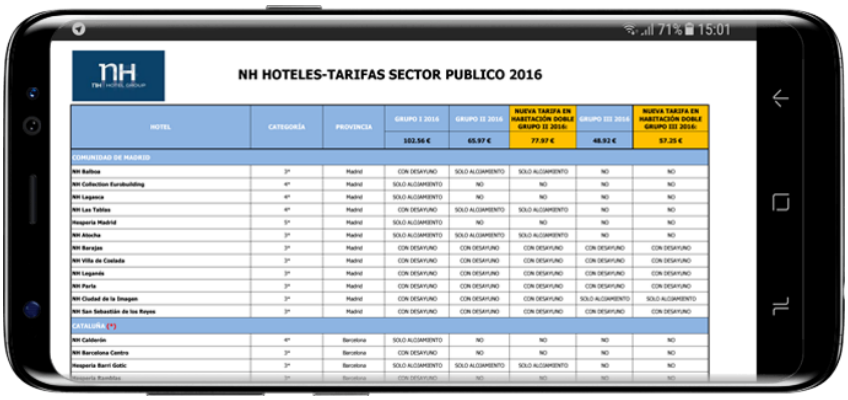


Imagen: Visualización de PDF con app instalada

- Si el cliente no dispone de ningún visor de archivos PDF instalado, al principio se le invitaba a que instalase una aplicación para tal fin en la tienda de aplicaciones de Android. Posteriormente se vio que esta no era una solución óptima, ya que afectaba negativamente a la experiencia de uso del cliente el hecho de descargar el archivo y encontrarse con un mensaje que le decía que no se podía visualizar y que se tenía que descargar otra aplicación.

  Por ello, continuamos investigando para ver cómo reducir la frustración en el usuario final y dimos con la clave. Implementamos una función para que, comprobase si el cliente tiene un visor instalado en su *Smartphone* y, en caso negativo, en vez de descargar el PDF lo que hará será abrirlo con el navegador del teléfono utilizando el visor online de *Google Drive*.

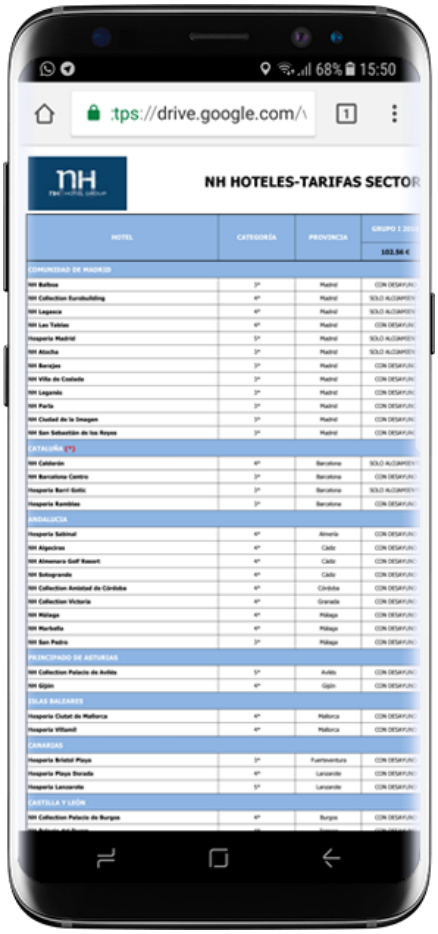


Imagen: Visualización de PDF sin app instalada

Este caso es cada vez menos posible que suceda, ya que actualmente la mayoría de terminales vienen con algún visor de archivos preinstalado de fábrica. Pero gracias a la opción del visor online de *Google Drive*, nos aseguramos de que en cualquier caso el usuario consiga ver el archivo, ya que este visor se abre con el navegador de Internet el cual es una aplicación que siempre viene preinstalada en el *Smartphone*.

Una vez implementada la visualización de archivos PDF, pasamos a implementar la **visualización de imágenes**. Los tipos de imagen que se soportan son: png, jpg, jpeg, gif, bmp y webp. Como se puede observar, se han abarcado una gran cantidad de formatos que soporta Android para que prácticamente cualquier imagen existente se pueda enviar y reproducir en el Smartphone. En cuando al formato webp, se lleva a cabo la comprobación de que la versión de SDK se al menos la 14 para que pueda soportarlo.

A diferencia de los archivos PDF, no se encuentra el problema de que el cliente no disponga de un visor de imágenes instalado, ya que cualquier Smartphone con cámara de fotos debe tener una galería para poder visualizar las fotografías. No obstante, también se lleva a cabo la comprobación de que se tenga instalada una aplicación para tal fin para así evitar posibles errores.

Al igual que en el caso anterior, si el cliente tiene instaladas varias aplicaciones para visualizar imágenes se le dará la opción de elegir con cual quiere abrirlas.

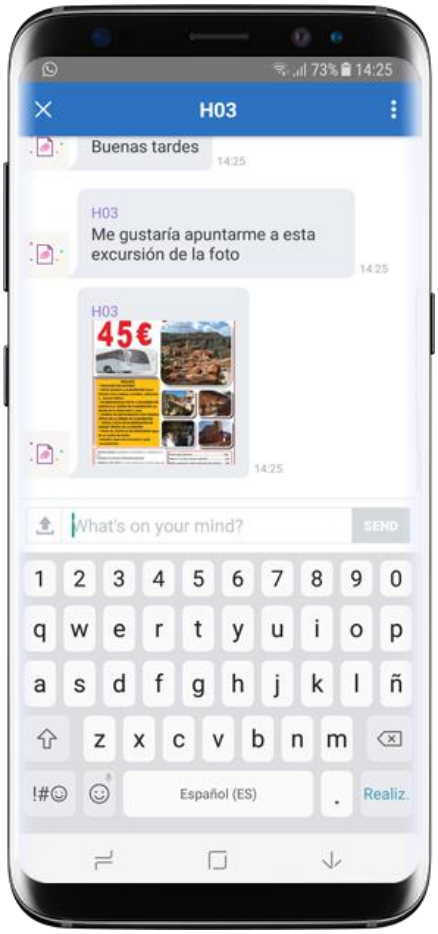


Imagen: Fotografía recibida

Imagen: Visualización de foto

Cuando se implementó la funcionalidad de recibir y visualizar imágenes, se pasó a añadir la misma utilidad con los vídeos. En este caso, los formatos que se soportan son: mp4, 3gp, webm y x-matroska. Para este último formato, x-matroska, se comprueba que la versión de SDK sea al menos la 14, ya que fue la versión a partir de la cual Android lo empezó a soportar.

Al igual que sucede en el caso anterior, aunque los *Smartphones* actuales disponen de una galería capaz de reproducir vídeos e imágenes, se lleva a cabo la comprobación de que dicha aplicación está instalada en el teléfono y en el caso de disponer de varias aplicaciones destinadas a tal fin, se da al usuario la posibilidad de elegir con cual quiere abrirla.

Imagen: Descarga de video

Imagen: Visualización de video

Interfaz web

Para diseñar la versión web del sistema de chat, las instrucciones por parte del CTO de *Sismotur* fueron claras. Éste quería una apariencia similar a la de *WhatsApp Web*, ya que, según él, presentaba un diseño simple y funcional.

Haciendo caso a estas indicaciones, se llevaron a cabo los siguientes pasos para diseñar la versión web:

1. Se optó por comenzar utilizando un fondo blanco acompañado de detalles en grises para dotar al chat del minimalismo que se buscaba. Para los títulos y características importantes del chat, fueron utilizados un color principal naranja y un color secundario azul característicos del logo de *Inventrip*.

2. En la parte izquierda de la aplicación web, donde se muestra la lista chats, se presentaba un reto por delante. Y es que, al ser un sistema de chat dedicado a hoteles, la vista utilizada en *WhatsApp Web* no era funcional ya que habría un número elevado de chats (uno por cada habitación del hotel más los canales grupales).

   Por esta razón, se propusieron varios diseños entre los miembros del grupo y finalmente se decidió aplicar el que se muestra en la imagen de la derecha:

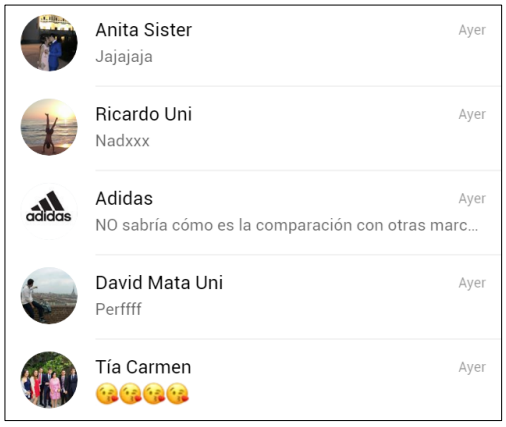


Imagen: Vista de chats en WhatsApp Web

Imagen: Solución propuesta

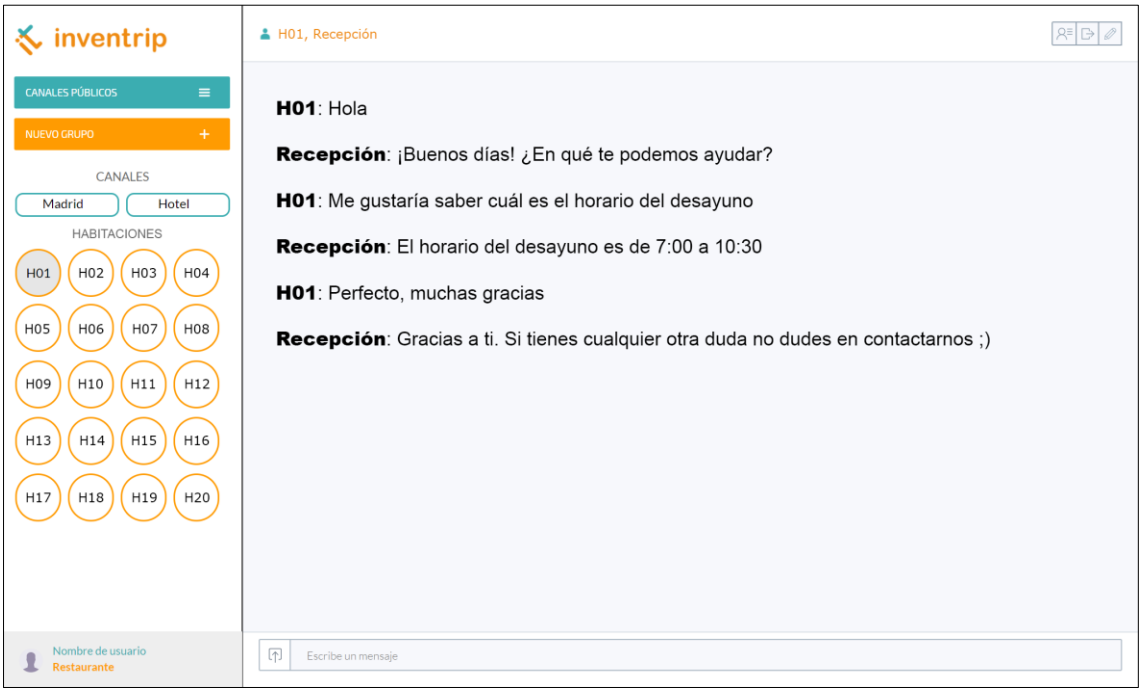
En este diseño, los chats de las habitaciones se encuentran representados por botones circulares con los números de las habitaciones. Así, se aprovechaba más el espacio y se conseguía visualizar un mayor número de chats. Para los canales grupales, los cuales son menos numerosos, se optó por situarlos en la parte superior introduciendo el nombre del canal en un diseño rectangular.

3. También se tuvo en cuenta que, al manejar un número elevado de chats, era necesario un sistema de notificaciones. Por ello, cuando llega un mensaje nuevo el empleado del hotel podrá ver en la lista de chats un globo con un contador de mensajes sin leer.

Además, se dispone de un sistema de notificaciones de escritorio, activable desde el navegador de Internet con el que el empleado del hotel podrá ver una notificación cada vez que llegue un mensaje independientemente de si se encuentra o no en la ventana de chats.

(Esta segunda forma de notificaciones iría mejor en implementación).

Así, el diseño final que se obtuvo fue el siguiente:



El CTO de *Sismotur* lanzó la propuesta de poder acceder al chat del hotel con los diferentes casos de usos mostrados en la sección de motivación del primer capítulo. Por ello, se diseñó una página intermedia desde la que el personal del hotel podría acceder como la recepción, el restaurante o el spa del hotel.

Implementación del servidor

Como se ha explicado en el apartado Tecnologías utilizadas, la integración de *SendBird* en un proyecto web es trivial. Tras copiar y pegar la *App ID* aportada por *SendBird*, ya teníamos acceso a la base de datos y las funciones de la API.

```
function startSendBird(userId, nickName) {
  sb = new SendBird({
    appId: appId
  });

  sb.connect(userId, function(user, error){
    if (error) {
      return;
    } else {
      initPage(user);
    }
  });
```

Como vemos en la imagen, *SendBird* utiliza la función *startSendBird()* para iniciar sb, con un usuario y nickname concreto, por lo que los datos obtenidos y modificados, serán guardados para el usuario concreto introducido.

En caso de fallo a la hora de establecer la conexión, no se avanzará a la página inicial.

Esta página inicial se encarga de cargar los chats, usuarios y canales de la base de datos asociada a nuestro usuario.

```
var initPage = function(user){
  isInit = true;
  $('.init-check').hide();

  currentUser = user;
  sb.updateCurrentUserInfo(nickName, '', function(response, error) {
    // console.log(response, error);
  });

  GroupChannelListQuery = sb.GroupChannel.createMyGroupChannelListQuery();
  OpenChannelListQuery = sb.OpenChannel.createOpenChannelListQuery();
  UserListQuery = sb.createUserListQuery();

  GroupChannelListQuery.limit = 100;
  GroupChannelListQuery.includeEmpty = true;
  OpenChannelListQuery.limit = 100;

  UserListQuery.limit = 100;

  getGroupChannelList();

  setTimeout(function(){
    updateGroupChannelListAll();
    setInterval(function(){
      updateGroupChannelListAll();
    }, 1000);
  }, 500);
};
```

Inicio de sesión

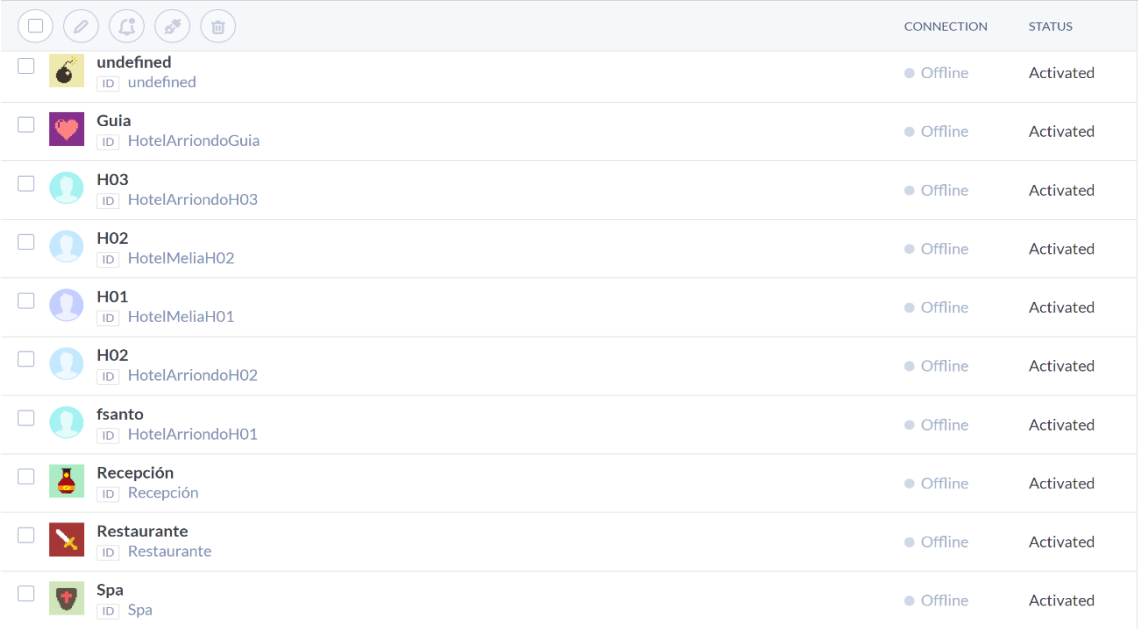
Nada más entrar en el chat, se comprueba si se ha iniciado sesión. Esta es una parte crucial para que no se desbloquee la aplicación en el caso de que se introduzca a mano en el navegador la ruta del chat.

Se comenzó por crear una vista de inicio de sesión, con un fondo típico de *Inventrip*. Aunque esta vista tenga menos importancia, se dividió en tres opciones:

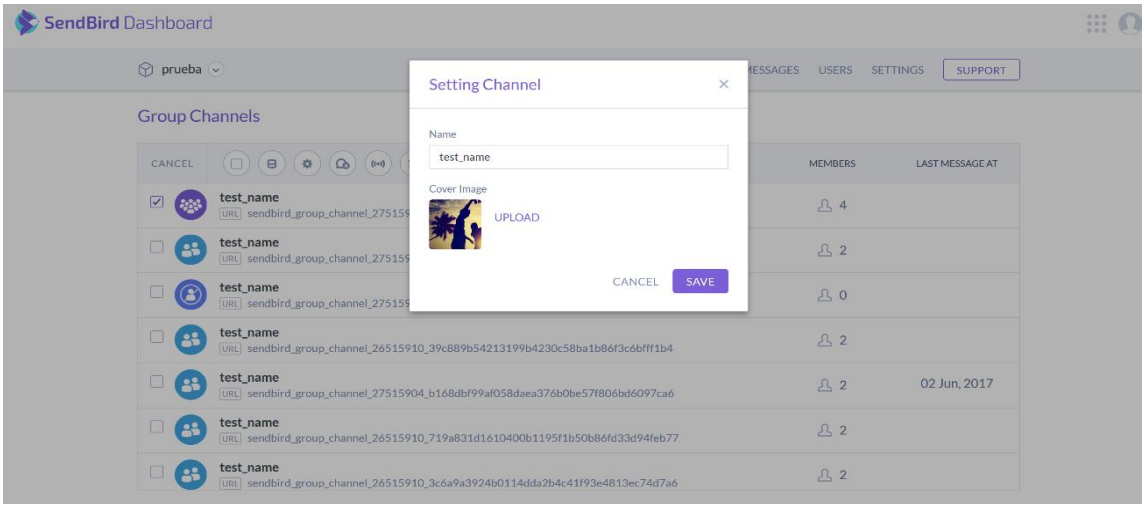
- Chat para Recepción
- Chat para Restaurante
- Chat para Spa

La idea original, se basaba en que el cliente podía iniciar sesión con cualquiera de esos 3 usuarios y de esta manera, utilizar el chat desde estos tres servicios, aunque el caso de uso se centra en uno genérico. Por lo tanto, al pulsar en cada uno de los botones, se inicia sesión con un usuario distinto.

Dado que la base de datos que ofrece *SendBird* es un poco limitada, ya que solo permite ver y clasificar los usuarios con un *UserName* y un *Nickname*, aquí encontramos el primer punto a tener en cuenta para pasarnos a utilizar otra tecnología como al final acabó siendo *ChatSDK* y *Firebase*.



Además, *SendBird* permite gestionar los mensajes de los grupos y canales a través del *dashboard*, aunque la mayor parte de las funcionalidades forman parte de la versión de pago, podemos realizar algunas modificaciones y visualizaciones en los datos de los grupos y canales, como puede ser cambiar el nombre o la foto del grupo, ver el número de usuarios que forman parte del chat, y la fecha de creación del mismo.



Flujo de entrada

El código del chat se estructura en base a métodos parametrizados reutilizables según diversos casos, como diálogos modales, elementos con clases idénticas. Se trata de ir tratando dinámicamente estos elementos, añadiendo o quitando clases según convenga, mostrando u ocultándolos, modificando su contenido HTML.

Flujo de salida

SendBird utiliza funciones *update()* y *refresh()* para mantener la base de datos y la aplicación actualizada. Resultan muy útiles a la hora de añadir o modificar elementos HTML en cuanto ocurra un cambio en la base de datos.

Usuarios

Los usuarios tienen las siguientes propiedades:

- UserName
- Nickname

El *UserName* es inmutable, y el encargado de obtener el usuario adecuado de la base de datos. Sin embargo, el campo *NickName* se puede modificar a gusto del usuario.

Canales públicos

Al pulsar el botón *Canales públicos*, se abre un desplegable modal a la derecha, que contiene una lista de los canales públicos de la base de datos.

Un canal público tiene las siguientes propiedades:

- Identificador único.
- Fecha de creación del canal
- Nombre del canal
- Imagen de perfil

Al seleccionar uno de la lista, se añade un elemento con el nombre al contenedor con canales públicos abiertos llamado CANALES mediante la función *addChannel()*. Esto consiste en crear un elemento HTML mediante el uso de una cadena de caracteres, con clases con apariencia definida en el documento CSS, y finalmente sustituir el contenido del elemento por los datos del canal.

Si se hace clic en uno de estos elementos, se entra en el canal público correspondiente, a través del método *joinChannel()*, mostrando la interfaz de su chat.

Dentro del canal público, el usuario puede:

- Editar el nombre del canal. Realiza una actualización para cambiar el nombre.
- Salir.

Posteriormente se explicará el envío y recepción de mensajes.

Canales grupales

Dado que los canales grupales permanecen en todo momento visibles, es necesario cargarlos nada más iniciar sesión. Para ello, se usó una consulta a la base de datos que devuelve los canales grupales del usuario en particular, en el método *getGroupChannelList()*. Este método recorre la lista de grupos (solo la clave) que está dentro del elemento usuario y llama a la función *addGroupChannel()*.

```
// Obtiene la lista de canales grupales (habitaciones, mesas...)
function getGroupChannelList() {
  GroupChannelListQuery.next(function(channels, error){
    if (error) {
      return;
    }

    channels.forEach(function(channel){
      var channelMemberList = '';
      var members = channel.members;

      channelMemberList = channelMemberList.slice(0, -2);
      addGroupChannel(true, channelMemberList, channel);
```

La función *addGroup()* se encarga de consultar la información completa del grupo, así como sus usuarios y añadir el elemento a la lista de grupos. Una vez más, se trata de ir recorriendo la consulta y añadir elementos HTML a una cadena de caracteres, insertando los nombres en cada elemento.

Un canal grupal tiene las siguientes propiedades:

- Identificador único
- Identificador del usuario que lo creó
- Fecha de creación
- Lista de usuarios en el grupo

El nombre del canal se obtiene a partir de los usuarios pertenecientes al mismo. Posteriormente, *Sismotur* podrá cambiar los nombres de los canales grupales por los nombres de las habitaciones.

Al seleccionar un elemento de la lista de grupos, se entra en el canal mediante la función *joinGroupChannel()*, de comportamiento similar a la análoga para canales públicos.

Dentro del chat, se muestran cuatro opciones:

- Invitar usuarios al chat. Muestra un desplegable con la lista de usuarios de la aplicación, usando *getUserList()*. Contiene además un filtro de búsqueda y un botón de invitar.
- Listar usuarios del chat. Muestra una lista con los usuarios que están en la habitación, usando *getMemberList()*.
- Eliminar el canal grupal. Llama a *borrarGrupo()*, que realiza una actualización para eliminar el canal posteriormente con *deleteChannel()*.

Mensajería

El método loadMoreChatMessage se encarga de realizar la consulta a la base de datos que devuelve todos los mensajes del chat actual. Posteriormente, recorre cada mensaje y lo añade al chat. Un mensaje contiene las siguientes propiedades:

- Date. Fecha de creación en milisegundos.
- Payload. Contenido del mensaje, ya sea un texto o una URL de la dirección de storage del archivo.
- Tipo. Texto (0), archivo (1) o imagen (2).
- Identificador del usuario que envió el mensaje.

Dentro de cada elemento mensaje, mostramos además del nombre del usuario y el contenido del mensaje, su fecha de envío y una opción de eliminar en el caso de que se trate de un mensaje propio.

Diálogos modales

Existen varios tipos de diálogos modales implementados en el chat:

- Diálogo de entrada *(modalInput())*. Se utiliza para obtener un dato de entrada.
- Diálogo de confirmación *(modalConfirm())*. Se utiliza para confirmar una acción, como por ejemplo eliminar un mensaje, salir de un canal o crear una nueva habitación.

Estos diálogos son reutilizables en su mayoría. Están parametrizados, pudiendo modificar el contenido del diálogo y permitiendo la posibilidad de que al confirmar se envíe una petición a una función *callback* pasada por el parámetro llamado *submit*.

Implementación con Chat SDK

Nada más terminar el desarrollo de la aplicación con la API *Sendbird*, se analizaron las conclusiones y se propusieron otros casos de uso, como por ejemplo desarrollar un sistema de planificación de viajes *offline*, que permitiese descargar un plan desde la nube y posteriormente poder abrirlo en local desde el dispositivo móvil. Estas ideas fueron descartadas de inmediato cuando Felipe Santi sugirió repetir el caso de uso del chat, con la diferencia de usar una API gratuita llamada Chat SDK y con la finalidad de realizar un análisis de ambas tecnologías posteriormente, y poner en distribución la mejor.

Interfaz móvil

En la versión *Android* de *Chat SDK* se utiliza un diseño dividido en 4 pestañas principales.

Las **dos primeras pestañas** son las más relevantes, ya que las componen los listados con los chats en los que podemos comunicarnos. Ambas pestañas presentan un diseño muy parecido a *WhatsApp* y aplicaciones similares de mensajería instantánea, por lo que será muy fácil que el usuario final se familiarice rápidamente a ellas.

- En la primera pestaña se encuentran agrupados los chats **uno a uno** y **grupales**. Los chats uno a uno son los más comunes ya que son los chats creados entre el cliente de una habitación y el hotel. Los chats grupales son un caso excepcional que lo componen el hotel junto con dos o más habitaciones en las que se hospedan personas conocidas entre ellas.

- En la segunda pestaña se muestran los **canales públicos**. Estos canales tienen el objetivo de difundir mensajes a un gran número de usuarios, por lo que es una importante herramienta para que los hoteles puedan enviar promociones a sus clientes.

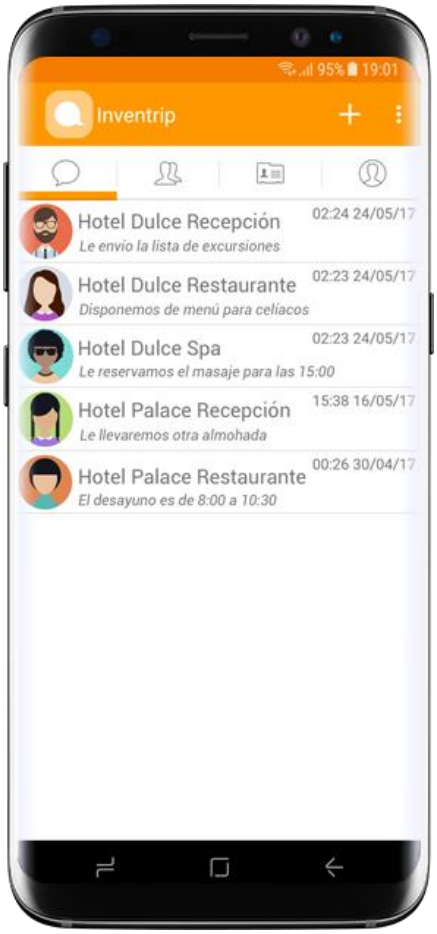


Imagen: Vista de chats uno a uno y grupales

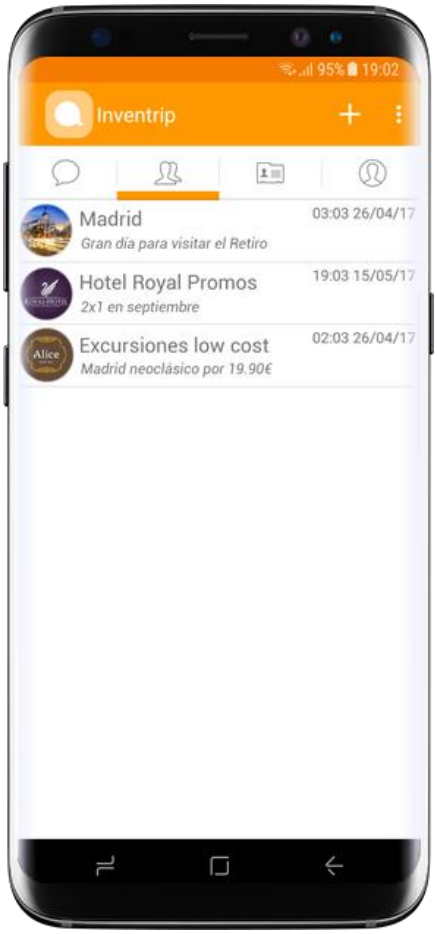


Imagen: Vista de canales públicos

En la **tercera pestaña** se encuentra la **lista de usuarios** que tienes agregados a tu lista de contactos. Esta ventana presenta un diseño muy similar a las anteriores pestañas de forma que la aplicación sigue resultando sencilla de manejar para el usuario final.

En esta pestaña, también se da la opción de poder **buscar un contacto** para comenzar a chatear. Se dispone de la opción de poder agregar varios contactos a la vez, ya que como se propusieron varios casos de uso (recepción, restaurante y spa) dependiendo de la zona del hotel con la que te quisieras comunicar, podrás agregarlos simultáneamente marcando los *check box* correspondientes.

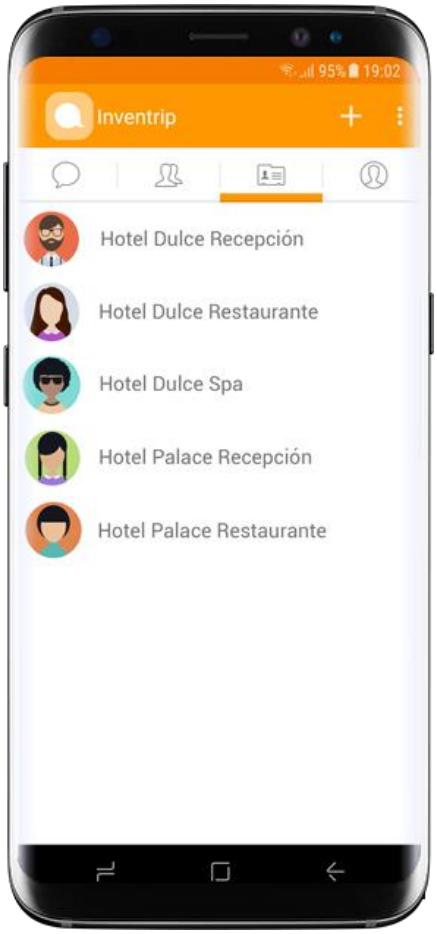


Imagen: Vista de usuarios agregados

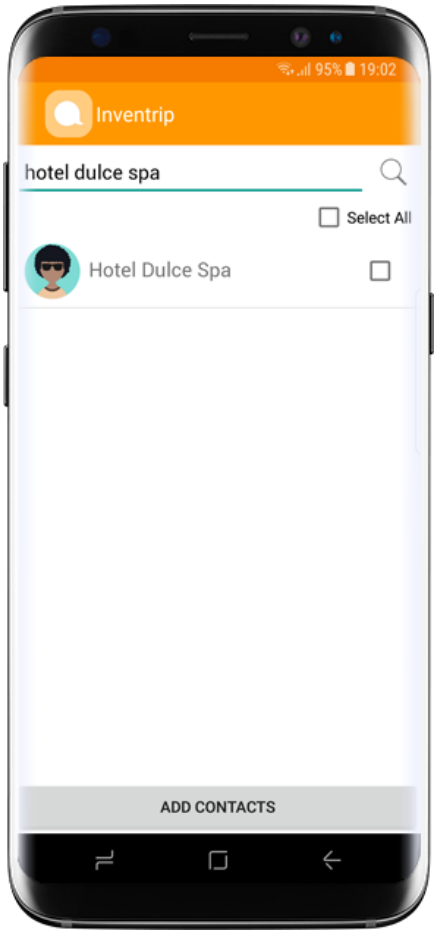


Imagen: Búsqueda y agregado de usuarios

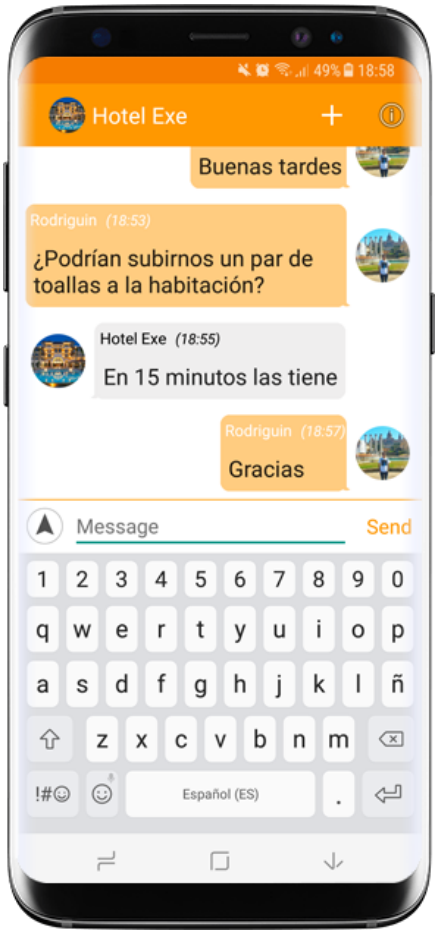
En la **cuarta y última pestaña** nos encontramos con la vista del perfil. En ella podemos modificar los siguientes campos:

- **Nombre personal**: útil para que el personal del hotel pueda dirigirse a la otra persona por su nombre de pila.
- **Número de teléfono**: campo opcional donde podremos introducir nuestro teléfono de contacto.
- **Correo electrónico**: útil para recuperar la contraseña de nuestra cuenta en el caso de olvidarla.

Además, desde esta pestaña podremos **cerrar la sesión** de nuestro usuario mediante el botón disponible en la parte superior derecha.

Imagen: Visualización y modificación de perfil.

Si se accede a una conversación, ya sea en un chat de grupo o en un canal, el diseño que se presenta es siguiente:



En este diseño se ha utilizado una tonalidad más clara de naranja para distinguir los mensajes escritos por el usuario y un fondo grisáceo para los mensajes recibidos por parte de la otra persona tal y como hacen las principales aplicaciones de mensajería instantánea.

En la cabecera de los mensajes, se muestra el nombre de usuario y la fecha en la que se envió el mensaje. Este nombre de usuario se especifica en la pestaña de perfil vista anteriormente y permite que el personal del hotel pueda dirigirse al cliente por su propio nombre.

Finalmente, en el resto del bocadillo se muestra el cuerpo del mensaje junto con la foto de perfil localizada en el lateral izquierdo o derecho dependiendo de si es el emisor o el receptor del mensaje.

Interfaz web

Debido a que ya se disponía de una interfaz para el chat desarrollada en la iteración anterior y, teniendo en cuenta la aprobación del CTO de *Sismotur*, el cual comunicó que le parecía perfecta, se decidió aprovecharla.

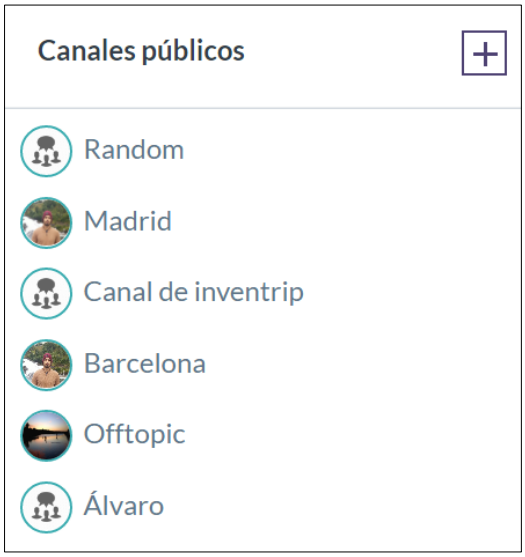
Es por ello que, aunque la implementación del chat con *Firebase* tuvo que hacerse otra vez desde cero, el diseño de la interfaz se mantuvo con cambios mínimos. No obstante, al continuar mejorando el chat con nuevas funcionalidades, se tuvieron que añadir algunos componentes a la interfaz.

Entre algunas de estas adaptaciones destacan:

- Ajustes en los **colores**. El naranja y el verde turquesa fueron usados para los botones de los diálogos modales, igual que para resaltar algunas palabras como el nombre del usuario en el mensaje del chat, siempre y cuando fuera el nombre del usuario registrado.
- Nuevos **botones** para dar soporte a nuevas funcionalidades descritas en la parte de implementación.
- Nuevas imágenes **SVG**.
- Nuevos diálogos **modales**.

También se modificó el diseño que representaba los botones de los chats de las habitaciones por unos rectangulares, siguiendo una línea de diseño más acorde con el resto de botones de la interfaz:

En cuanto a las imágenes de perfil de usuarios y grupos, también se actualizó su diseño por uno circular más moderno tal y como vienen haciendo las principales aplicaciones de mensajería en los últimos años.



Implementación del servidor

Tras haber elegido la API de Chat SDK como mejor opción, se encontró el problema de que esta API no contaba con soporte web, pero dado que estaba implementaba sobre Firebase se decidió hacer una implementación web con el mismo diseño de la aplicación web implementada sobre Sendbird, pero utilizando Firebase y adaptándonos a la implementación dada por Chat SDK.

En este punto se procederá a explicar en qué consistió la integración de *Firebase* con la interfaz de chat montada en la anterior iteración.

Integración

Como se ha explicado en el apartado *Tecnologías utilizadas*, la integración de *Firebase* en un proyecto web es trivial. Tras copiar y pegar la variable de configuración con la información de la base de datos, se creó una instancia de la misma, inicializando la API.

Con *Firebase.auth()* se inicializó un objeto de usuario autenticado en la aplicación. Este objeto contiene datos de interés como el nombre de usuario, su identificador único, o su email.

Con *Firebase.database* se puede controlar todos los métodos y consultas de la base de datos. Para acceder a los archivos e imágenes, se necesita otra instancia, *Firebase.storage*.

Inicio de sesión

Se comenzó por crear una vista de inicio de sesión, con un fondo típico de *Inventrip*. Aunque esta vista tenga menos importancia, se dividió en tres opciones:

- Chat para **Recepción**
- Chat para **Restaurante**
- Chat para **Spa**

En un principio, la idea constaba de poder utilizar el chat desde estos tres servicios, aunque el caso de uso se centra en uno genérico. Por lo tanto, al pulsar en cada uno de los dos botones, se muestra un diálogo modal de inicio de sesión, registro y recuperación de contraseña.

Cada uno de estos tres botones tiene asignado un evento que envía los datos de los campos a *Firebase*, utilizando los siguientes métodos:

- **signInWithEmailAndPassword**. Intenta iniciar sesión con las credenciales obtenidos de los dos campos de texto.
- **createUserWithEmailAndPassword**. Registra un nuevo usuario en la aplicación.
- **sendPasswordResetEmail**. El mensaje enviado se puede modificar desde *Firebase console.*

Firebase cuenta con un sistema de gestión de errores, que fue aprovechado para informar en cualquier tipo de error al usuario, dentro del propio diálogo modal, en la parte inferior.

No se decidieron implementar todas las posibilidades de *Firebase.auth()*, ya que el proyecto se centra en el propio chat. Posteriormente, *Sismotur* podrá adaptar este código a sus necesidades.

Configuración del chat

Nada más entrar en el chat, se usó el método *Firebase.auth().onAuthStateChanged* para comprobar si se ha iniciado sesión. Esta es una parte crucial para que no se desbloquee la aplicación en el caso de que se introduzca a mano en el navegador la ruta del chat.

Una vez comprobada la autenticación, se llama al método *init()*, donde se carga la información del usuario identificado y sus canales grupales (habitaciones del hotel).

Flujo de entrada

El código del chat se estructura en base a métodos parametrizados reutilizables según diversos casos, como diálogos modales, elementos con clases idénticas. Se trata de ir tratando dinámicamente estos elementos, añadiendo o quitando clases según convenga, mostrando u ocultándolos, modificando su contenido HTML.

Mediante *callbacks*, se envían los datos a métodos de gestión de los mismos, encargados de enviar las actualizaciones a *Firebase.*

Flujo de salida

Aprovechando la asincronía de *Firebase*, se ha prestado atención a los eventos *value* y *child_added* del método *on()*. Resultan muy útiles a la hora de añadir o modificar elementos HTML en cuanto ocurra un cambio en la base de datos.

Usuarios

Los usuarios son el elemento central de la aplicación ya que a partir de ellos se obtienen todos los datos necesarios para cargar los diferentes chats.

Los usuarios tienen las siguientes propiedades:

- Identificador único.
- Ultima vez online
- E-mail
- Nombre
- Imagen
- Teléfono
- Lista de chats grupales (identificadores de grupo)

Dentro de la aplicación se podrán modificar los valores de e-mail, nombre, imagen y teléfono a través de la función *editarUsuario()*. Además, se usará la lista de chats grupales para cargar la lista de grupos al inicio de la aplicación como se explica más adelante.

Canales públicos

Al pulsar el botón Canales públicos, se abre un desplegable modal a la derecha, que contiene una lista de los canales públicos de la base de datos.

Un canal público tiene las siguientes propiedades:

- Identificador único.
- Identificador del usuario que lo creó
- Nombre del canal
- Foto de perfil
- Tipo de canal (0 privado, 1 público)

Para listar los canales, se realizó una consulta a *Firebase* preguntando por los canales, de los cuales se cogieron los de tipo público (1) y se fueron añadiendo uno a uno a la lista con *prependChannel()*, creando un elemento que contiene el nombre del canal y la imagen de perfil.

Al seleccionar uno de la lista, se añade un elemento con el nombre al contenedor con canales públicos abiertos llamado *CANALES* mediante la función *addChannel()*. Esto consiste en crear un elemento HTML mediante el uso de una cadena de caracteres, con clases con apariencia definida en el documento CSS, y finalmente sustituir el contenido del elemento por los datos del canal.

Si se hace clic en uno de estos elementos, se entra en el canal público correspondiente, a través del método *joinChannel()*, mostrando la interfaz de su chat.

Dentro del canal público, el usuario puede:

- Editar el nombre del canal. Realiza una actualización a *Firebase* para cambiar el nombre del canal, y se actualiza en el contenedor.
- Salir, lo que eliminaría el elemento del contenedor *CANALES*.

Posteriormente se explicará el envío y recepción de mensajes.

Canales grupales

Dado que los canales grupales permanecen en todo momento visibles, es necesario cargarlos nada más iniciar sesión. Para ello, se usó una consulta a la base de datos que devuelve los canales grupales del usuario en particular, en el método *getGroupChannelList()*. Este método recorre la lista de grupos (solo la clave) que está dentro del elemento usuario y llama a la función *addGroup(grupo_id)*. Además, esta

función añade un *listener* a la lista de grupos del usuario para recibir si se añade a algún grupo al usuario.

La función *addGroup(grupo_id)* se encarga de consultar la información completa del grupo y añadir el elemento al contenedor *GRUPOS*, además añade un *listener* que recibe cualquier cambio del grupo como añadir un usuario o recibir un mensaje. Una vez más, se trata de ir recorriendo la consulta y añadir elementos HTML a una cadena de caracteres, insertando los nombres en cada elemento.

Un canal grupal tiene las siguientes propiedades:

- Identificador único
- Identificador del usuario que lo creó
- Tipo de canal (0 privado)
- Fecha de creación
- Lista de usuarios en el grupo

El nombre del canal se obtiene a partir de los usuarios pertenecientes al mismo, ya que en la API *ChatSDK*, se hace de esta manera. Posteriormente, *Sismotur* podrá cambiar los nombres de los canales grupales por los nombres de las habitaciones.

Al seleccionar un elemento de *GRUPOS*, se entra en el canal mediante la función *joinGroupChannel()*, de comportamiento similar a la análoga para canales públicos.

Dentro del chat, se muestran cuatro opciones:

- Invitar usuarios al chat. Muestra un desplegable con la lista de usuarios de la aplicación, usando *getUserList()*. Contiene además un filtro de búsqueda y un botón de invitar.
- Listar usuarios del chat. Muestra una lista con los usuarios que están en la habitación, usando *getMemberList()*.
- Eliminar el canal grupal. Llama a *borrarGrupo(),* que realiza una actualización a *Firebase* para eliminar el canal posteriormente con *deleteChannel()*.

Mensajería

El método *loadMoreChatMessages()* se encarga de realizar la consulta a la base de datos que devuelve todos los mensajes del chat actual. Posteriormente, recorre cada mensaje y lo añade al chat. Un mensaje contiene las siguientes propiedades:

- Date. Fecha de creación en milisegundos.
- Payload. Contenido del mensaje, ya sea un texto o una URL de la dirección de *storage* del archivo.
- Tipo. Texto (0), archivo (1) o imagen (2).
- Identificador del usuario que envió el mensaje.

Según el tipo de mensaje, se añade al chat con un método u otro. Como se puede observar, el mensaje no contiene el nombre del usuario. Para obtener el nombre de usuario en un canal privado, basta con mirar en el objeto *users* del canal actual obtenido en la consulta. Si se trata de un canal público, este objeto no se encuentra en la base de datos, por lo que hacemos un producto cartesiano con el objeto *userList* obtenido al iniciar sesión. Esto se describirá posteriormente en la sección *Problemas encontrados*.

Para recibir mensajes, usamos el evento *child_added* del método *on()* de *Firebase*. Esta función se activa cada vez que se añade un hijo a un lugar específico de la base de datos. Nos permite crear un nuevo elemento HTML mensaje y añadirlo al chat.

Dentro de cada elemento *mensaje*, mostramos además del nombre del usuario y el contenido del mensaje, su fecha de envío y una opción de eliminar en el caso de que se trate de un mensaje propio.

Por último, se ha añadido la posibilidad de enviar mensajes difundidos a los canales grupales. Esta funcionalidad cumple el requisito de querer enviar publicidad a las habitaciones del hotel. Para ello, se añadió un icono en el contenedor *GRUPOS*, que abre un diálogo modal donde se toma el texto. Al enviar el mensaje, se recorren todos los canales grupales del usuario y se envía una actualización a la base de datos con el mensaje nuevo.

Diálogos modales

Existen varios tipos de diálogos modales implementados en el chat:

- Diálogo de **sesión** (*modalLogin()*). Se utiliza en el inicio de la aplicación.
- Diálogo de **entrada** (*modalInput()*). Se utiliza para obtener un dato de entrada.
- Diálogo de **confirmación** (*modalConfirm()*). Se utiliza para confirmar una acción, como por ejemplo eliminar un mensaje, salir de un canal o crear una nueva habitación.
- Diálogo **multimodal** (*multiModalInput()*). Se utiliza para gestionar la información de perfil del usuario autenticado.
- Diálogo de **difusión** (*modalBroadcast()*). Se utiliza para enviar un mensaje difundido a los canales grupales.

Estos diálogos son reutilizables en su mayoría. Están parametrizados, pudiendo modificar el contenido del diálogo y permitiendo la posibilidad de que al confirmar se envíe una petición a una función *callback* pasada por el parámetro llamado *submit*

Capítulo V
EVALUACIÓN

Problemas encontrados

Temática del TFG

Al principio, el TFG propuesto consistía en el desarrollo de una aplicación para iOS dedicada al turismo en Ibiza. Pero durante la primera reunión del curso, el CTO de *Sismotur* nos comunicó que esa idea ya la habían desarrollado durante el verano por exigencias de la empresa, ya que querían sacarla al mercado cuanto antes.

Por esta razón se propuso la alternativa de actualizar el código de la aplicación existente en iOS a la última versión de Swift 3. Pero nos encontramos con el problema de que sólo se disponía de un Mac Mini no lo suficientemente potente.

Finalmente, la empresa nos ofreció realizar el trabajo sobre otra idea que tenían en mente; realizar un sistema de comunicación por chat dedicado exclusivamente a hoteles.

Incompatibilidad entre diferentes versiones de Android

Uno de los problemas que se encontraron, fue debido a la incompatibilidad entre las diferentes versiones de *Android* existentes en el mercado. En concreto, queríamos ampliar nuestro sistema de chat incluyendo funciones que permitieran visualizar imágenes, vídeos y archivos PDF recibidos en el móvil, por lo que necesitábamos acceder al almacenamiento interno del teléfono.

Para implementar estas funciones, pasábamos la URI de la forma *"file://..."* y funcionaba correctamente en todas las versiones excepto en la última disponible en el emulador de *Android Studio (Android 7 Nougat)*. Se vio que desde la versión de SDK 24 en adelante, se dejó de dar soporte a esta implementación por temas de seguridad.

Por esta razón se decidió cambiar la implementación utilizando *File Provider*. Así se consiguió que funcionara el *Android 7 Nougat*. Sin embargo, probándolo con *Android 4.4 Kit Kat*, una de las versiones a la que nuestro sistema de chat debía de dar soporte, se vio que por su antigüedad aun no soportaba el uso de *File Provider*. Es decir, en las versiones más nuevas no se podía utilizar una implementación y en las versiones más antiguas no se podía utilizar la otra.

Finalmente, la solución pasó por combinar ambas implementaciones dependiendo de la versión de *Android* del dispositivo utilizado.

Nuevos formatos de pantalla ultra panorámicos

Durante el pasado mes de abril de este año, han sido lanzados al mercado nuevos modelos de Smartphone que no siguen el formato de pantalla 16:9 al que venimos estando acostumbrados. Estos modelos son el LG G6 y los Samsung Galaxy S8 y S8 Plus, los cuales tienen formatos de pantalla 18:9 y 18’5:9 respectivamente. Se vio que todas las aplicaciones necesitaban actualizarse para visualizarse de forma correcta en estos formatos de pantalla.

Por ello, se buscó la solución para dar soporte a los futuros usuarios del hotel que tuvieran estos modelos de Smartphone. En este caso, la solución fue añadir unas líneas de código al manifiesto de la aplicación y así nuestra aplicación ya se visualiza correctamente en estos nuevos formatos de pantalla.

Imagen: LG G6

Imagen: Galaxy S8

SendBird: bases de datos limitadas

Uno de los problemas que nos surge en la primera iteración está relacionado con la base de datos, la base de datos que nos ofrece *SendBird* es muy limitada, los usuarios solo disponen de 2 campos como ya se ha comentado anteriormente, lo cual no nos permitía realizar ciertas funciones, encontrábamos soluciones para realizarlas, pero eran poco eficaces. Concretamente en la búsqueda y filtrado de los usuarios en función del Hotel en el que se encontraban, ya que los chats no permitían una comunicación “privada” entre los clientes del hotel. Los cuales podrían comunicarse con usuarios de otros Hoteles.

SendBird: Cambio de sesión de un usuario

Otro problema que se nos planteaba, era la forma en la que los usuarios se registrarían y serian desconectado de la aplicación, dado que los usuarios son un número fijo, como podría ser un usuario por habitación (50 habitaciones = 50 usuarios), cada vez que un cliente nuevo llegara a la habitación, debería automáticamente desconectar al anterior. Discutimos esto durante varias semanas y se nos ocurrieron alternativas que podrían servir para una futura implementación, entre las que se encontraban utilizar códigos QR, y una un tanto más complicada y ambiciosa, como era relacionar la propia aplicación con la de gestión de reservas del hotel, de modo que cuando un cliente hiciera *check-in* en una habitación, se le crearan automáticamente los chats con los principales usuarios (Recepción) y de forma inversa al realizar el *check-out*, se les desconectara automáticamente de la aplicación y así no pudieran seguir contactando con el Hotel, manteniendo así la privacidad de los usuarios.

Firebase: consultas anidadas

Debido a que *Firebase* es una base de datos síncrona, las consultas se realizan en segundo plano. Esto supone un problema a la hora de usar consultas anidadas, ya que, en la mayoría de los casos, se van a tratar variables que no tienen datos asignados en momentos inoportunos. *Firebase* devuelve objetos tipo *Promise*, por lo que se debe respetar este flujo.

La primera solución implementada fue la creación de un método recursivo de obtención de mensajes completos, que realizaba consultas anidadas para obtener el nombre de los usuarios en los canales públicos. Esta solución fue rápidamente descartada debido a su alto coste de cómputo. Cargar quince mensajes necesitaba un tiempo de tres segundos.

La mejor solución a este problema se explica en el siguiente punto.

Firebase: nombres de usuario en canales públicos

Como se ha mencionado en el apartado de implementación sobre *Firebase*, la implementación de *ChatSDK* requiere una determinada estructura de la base de datos. En esta estructura, los canales públicos no almacenan el nombre de los usuarios. Esto conllevaba un problema a la hora de mostrar el nombre del usuario de cada mensaje.

La primera solución que se planteó fue añadir una sección de usuarios a los chats públicos, pero esto requería modificar la implementación de *ChatSDK*, de una complejidad importante. Al intercambiar algunos correos electrónicos con los creadores de esta API, se llegó a la conclusión de que no merecía la pena que ya se necesitarían varias sesiones por teléfono para comprender toda la estructura del código.

La segunda solución, en un principio, poco ortodoxa, resultó ser la más adecuada. Cuando el usuario hace clic en el desplegable *NUEVO GRUPO*, se abre un desplegable con la lista de usuarios de la aplicación. Esto se logra mediante una consulta a *Firebase*. Estos datos son de la forma *idUsuario: nombreUsuario*, por lo que se decidieron aprovechar, obteniéndolos en el inicio de la aplicación.

Una vez dentro de cualquier canal, se usa esta tabla para conseguir los nombres de los usuarios, en cualquier caso: tanto en canales públicos como en privados.

Análisis

Este capítulo se centrará en un análisis comparativo desde el punto de vista de la empresa *Sismotur*, quien pondrá en producción el sistema de chat.

SendBird

Ventajas

A simple vista, *SendBird* presenta una amplia gama de posibilidades. Las más significativas son las siguientes:

- Bot de chat. La posibilidad de poder obtener información de manera instantánea, como puede ser el menú del día del hotel o los horarios del spa, genera un interés muy alto para la empresa y para el hotel, ya que ahorra tiempo y trabajo a los recepcionistas.
- Push notifications. Un turista puede no querer tener que estar atento al teléfono durante el día para comprobar si se le ha respondido a su pregunta. Con este sistema, la postura del cliente se reduce a usar la aplicación cuando reciba una notificación.
- Admin messages. Es una forma llamativa de priorizar mensajes de recepción, sobre temas importantes relativos al hotel, como emergencias o publicidad.
- Spam flood protection. Para impedir que un usuario sature los canales públicos, se podría usar esta funcionalidad.
- Auto translation. Este aspect es de los más importantes para la aplicación, ya que un hotel puede hospedar clientes de todas partes del planeta.

Desventajas

Sin embargo, *SendBird* no es una herramienta gratuita.

- Número limitado de usuarios de forma gratuita.

- Los mensajes se almacenan en la base de datos durante **tres meses** como máximo. A priori no puede significar un problema, pero todo depende de los intereses del hotel.
- El límite de datos enviados al mes es de 5GB.
- El tamaño máximo de un archivo que se puede enviar es de 5MB. Puede suponer un problema para el turista y para el hotel.

En general, la mayoría de funcionalidades de interés son solamente accesibles a través de paquetes *premium*, que cuestan a partir de 600€. El mantenimiento y consultoría de la API también tiene un coste, que depende de la aplicación.

Firebase

Con todas las desventajas que presentó *SendBird*, se decidió usar esta novedosa API con el fin de comparar ambas y poner en producción el chat con una de las dos.

Ventajas

Según se entra en el *Dashboard* de *Firebase*, se pueden observar muchas características llamativas:

- **Inicio de sesión** con varias cuentas. Permite registrarse con *Facebook*, *Twitter*, *GitHub*, *Gmail*, e incluso de forma anónima. Esta última presenta una ventaja significativa a la hora de querer entrar en la aplicación para preguntar algo en uno de los canales públicos.
- El ***Dashboard*** resulta muy manejable y cómodo a la hora de gestionar cualquier información de la aplicación, ya sea la base de datos, o la configuración de los correos de restablecimiento de contraseña. No se ha tenido ningún problema a la hora de entender cada una de ellas.
- Base de datos **síncrona**. Al principio ha costado acostumbrarse a las consultas en segundo plano. Sin embargo, la mayoría de las tecnologías hoy en día, como *NodeJS*, presentan esta característica, lo que nos ha ayudado a entender el flujo de datos y el concepto de *Promise*. La base de datos se actualiza de forma instantánea, y no se ha tenido que usar ninguna función periódica como puede ser *setInterval()*.
- El **código es muy abierto**. Además, la documentación es muy completa, ya que ofrece ejemplos para cada uno de los casos de uso de las funcionalidades de la base de datos, autenticación y almacenamiento.
- Ofrece la posibilidad de **hosting**. Es un aspecto que se evaluará en *Sismotur*.
- También facilita poner anuncios en la aplicación con fines monetarios.
- Todas las funcionalidades son gratuitas a pequeña escala. A mayor escala, depende de la cantidad de usuarios que exista en la aplicación. El **coste total de propiedad**, en este caso, sería de 200€ al mes, si la aplicación almacena más de 100.000 usuarios.

- El **mantenimiento**, por otro lado, se realiza mediante la gestión del *Dashboard*, y como se ha comentado anteriormente, resulta ser una tarea sencilla.

Desventajas

- No presenta sistema de notificaciones.
- No presenta sistema de traducciones y algunas funcionalidades *premium* de *Sendbird* como el bot, o los mensajes de prioridad.

Comparativa

Desde el punto de vista de la empresa, *Firebase* parece ser la opción más viable. No solo es más barata, sino que también presenta posibilidades ilimitadas si se desean. La cantidad de funcionalidades que se pueden implementar con el uso de las funciones proporcionadas depende de las necesidades e imaginación de la empresa, pero no se han presentado barreras.

Unas cuantas funcionalidades no gratuitas en *Sendbird* se han conseguido implementar con *Firebase* de manera no muy compleja:

- Mensajes difundidos de publicidad. Actualmente ya existe esta posibilidad en el chat, y podría evolucionar para tener un amplio abanico de opciones, como enviar a un determinado grupo de habitaciones, o a los usuarios suscritos a un servicio.
- Filtros.

Algunas de las ventajas que presentaba *SendBird*, se pueden implementar de forma gratuita con *Firebase*, como pueden ser:

- Sistema de traducciones de mensajes. Por medio de *Google Translation API*, se puede conseguir la misma funcionalidad sin necesidad de pagar una cuota mensual.
- La base de datos en formato *JSON*, ofrece infinidad de posibilidades de configuración, por lo que se podrían implementar mensajes prioritarios o el bot.
- Notificaciones. Añadiendo *listeners*, se podría obtener el mismo efecto que los mensajes sin leer en la interfaz de *Sendbird*, y así mostrar una forma cómoda de saber si alguna habitación ha pedido un servicio o ha hecho alguna pregunta.

Capítulo VI

CONCLUSIONES

Resumen

Tras la comparación entre *SendBird,* y por otro lado *Firebase* con ayuda de *ChatSDK*, la opinión del grupo es que ha sido más fácil y versátil de trabajar con la segunda iteración. Nos ha planteado retos durante días, que hemos sido capaces de superar.

En términos de comodidad, *Firebase* resalta en este aspecto, ya que su flexibilidad y facilidad de administración de la consola (*Dashboard*) y su amplia documentación han hecho el trabajo más llevadero. Además, *Firebase* es fácilmente escalable en torno al número de usuarios, y pagar 200€ mensuales no supone ningún problema para una empresa medianamente grande.

También facilita la adaptación del código en iteraciones futuras, ya que la base de datos es un archivo *JSON* que se puede importar y exportar con facilidad.

Competencias adquiridas

Durante el desarrollo de este trabajo de fin de grado, hemos adquirido una serie de conocimientos, tanto tecnológicos como relacionados con el mundo laboral.

En cuanto a las tecnologías utilizadas, cabe destacar:

- Desarrollo de aplicaciones web en JavaScript. Aunque se ha dado este lenguaje en asignaturas anteriores, se ha conseguido llegar a una limpieza y organización en el código muy superior a la de proyectos anteriores. Al haber utilizado APIs como *SendBird*, se ha podido observar la estructura de un código implementado por profesionales, y se ha adquirido esa mecánica de programación para el resto del proyecto. *Firebase* ha facilitado la comprensión de formatos de objeto como *JSON*, y se ha especialmente adquirido la capacidad de programar en segundo plano, con una base de datos asíncrona en tiempo real.

- Desarrollo de aplicaciones móviles. Para ello, el entorno de desarrollo utilizado ha sido Android Studio, el cual algunos miembros del grupo desconocíamos y hemos aprendido a manejar en profundidad. También hemos logrado aprender a solucionar los fallos que nos surgían durante la utilización de esta plataforma y el uso de las API anteriormente nombradas.

En cuanto al mundo laboral, el CTO de Sismotur nos ha enseñado a ver cómo funciona la empresa, a conocer las últimas novedades que surgían en el mercado, a descubrir que el primer camino no era siempre el definitivo y había que explorar varias posibilidades hasta dar con la mejor solución

Trabajo futuro

Ahora que se ha concluido con el desarrollo de la herramienta descrita en este Trabajo de Fin de Grado, no todo acaba aquí. Esta sección tiene como objetivo describir las expectativas de trabajo futuro sobre este proyecto.

Implementaciones adicionales

Por falta de tiempo, no se ha podido implementar el sistema de traducciones con *Google Translation API*. Para poder integrar esta funcionalidad, se tendría que montar el código sobre la librería *NodeJS*. Otras ideas posibles para la mejora del chat serían:

- Implementar un sistema de **notificaciones** de mensajes no leídos en los chats grupales. Esta es una de las características implementadas en *SendBird* que no se ha podido llevar acabo con *Firebase* por falta de tiempo. Sin embargo, no es una tarea costosa. Se activaría un *listener* cuando uno de los canales grupales sufra una modificación en la base de datos, y posteriormente, con ese identificador, se colocaría un contador de mensajes no leídos en el recuadro correspondiente de la habitación. Se crearía un elemento HTML para mostrar u ocultar este valor.
- Añadir clasificaciones de los canales grupales por **suscripciones** a servicios, como puede ser el spa o excursiones. Se tendría que modificar la estructura de la base de datos de *Firebase* para añadir un campo con el nombre de la suscripción, y posteriormente ir comprobando las suscripciones de cada canal grupal. De esta manera, se podría tener una clasificación de servicios en los que poder mandar publicidad y mensajes difundidos.
- **Adaptar el código de *ChatSDK*** para mantener la coherencia con estas nuevas implementaciones. Se tendrán que implementar métodos de escucha de eliminación de mensajes y de canales grupales, ya que es una característica que la API no presenta por defecto. Para ello, se hablará con los creadores de *ChatSDK* para modificar el código y generar una aportación al proyecto de *Github*.
- Terminar la implementación del **restablecimiento** de contraseña. Es una funcionalidad en la que no se ha hecho hincapié, debido a que se desviaba demasiado del caso de uso principal.
- Agregar un **bot** al chat. Es una característica que muchos chats actuales poseen. Ahorraría mucho tiempo a la recepción del hotel a la hora de contestar preguntas comunes y distribuir publicidad de forma automática.

El siguiente paso, sería adaptar el código a una escala mayor, con la posibilidad de almacenar más de un hotel en la base de datos de *Firebase*. Para realizar esto, basta con modificar la estructura de las reglas, y adaptar las consultas para que atraviesen un nivel más. Al iniciar sesión, se pediría el identificador del hotel, y todas las consultas internas se realizarían en torno a ese identificador.

Despliegue en pruebas de la herramienta

El siguiente paso, sería realizar el despliegue del sistema de chat por los hoteles. Para ello, lo primero que se necesitaría sería un hotel o un número reducido de ellos para empezar a realizar el despliegue en modo de pruebas. Este despliegue en modo de pruebas serviría para ver lo que funciona correctamente y lo que podría ser mejorado, hasta converger hacia un sistema satisfactorio para el hotel y sus clientes.

Otra de las cosas que se pondría en práctica, sería el uso real de los *beacons*. La idea original sería colocar un *beacon* en cada habitación del hotel como puerta de acceso al chat entre la habitación y el hotel. Es decir, el cliente llegaría a su habitación, se conectaría al chat, y la aplicación detectaría el beacon de la aplicación automáticamente de manera que su interlocutor supiera en dónde se encuentra (en el spa, restaurante o habitación). Esto sería posible regulando la intensidad de los *beacons* para que no interfieran entre ellos, es decir, una persona no debería poderse conectar al *beacon* de la habitación de al lado.

Otras alternativas propuestas a los *beacons* son los códigos QR o la tecnología NFC. Los códigos QR son una opción más económica a los *beacons*, ya que únicamente se necesitaría colocar un papel en la mesilla de cada habitación con el código QR. La tecnología NFC es una solución al problema del alcance de los *beacons* desde diferentes habitaciones, ya que se necesita acercar el Smartphone a una distancia máxima de 20 cm.

Comparando estas tecnologías, la mejor opción son los *beacons*, ya que basta con que el cliente active el *Bluetooth* del teléfono y ya no se tiene que preocupar de escanear códigos o estar a distancias concretas.

Despliegue final de la herramienta

Cuando ya se haya concluido con el despliegue en pruebas de la herramienta y se hayan (i) detectado y solventado los posibles problemas que pudieran surgir en su puesta en marcha, y (ii) asegurado la usabilidad y la utilidad del sistema por parte de los clientes y gerentes del hotel, el siguiente paso sería desplegarlo definitivamente por un mayor número de hoteles.

Para ello se necesitaría una buena cartera de hoteles en los que llevar a cabo el despliegue. *Sismotur*, al ser una empresa de turismo, podría conseguir fácilmente un buen número de clientes.

Si este sistema funciona en los hoteles de España, la visión de futuro sería distribuirlo también por Europa y por el resto del mundo.

Desarrollo en iOS

Otra posibilidad de trabajo futuro, sería la realización de una versión de la aplicación para iOS. El Trabajo de Fin de Grado propuesto en principio iba a ir orientado al desarrollo de otra aplicación para iOS, pero finalmente fue cancelado por problemas de disposición de un dispositivo con una versión de OS X que soportase la programación de Swift 3 mediante Xcode.

Por lo que esta sería otra línea a partir de la cual se puede trabajar en el futuro. Se podría realizar primero la prueba de la aplicación en Android (el cual tiene una cuota de mercado de un 92,8% aproximadamente en España), y viendo los resultados obtenidos, proceder al desarrollo en iOS.

Summary

After comparing *SendBird*, and on the other hand *Firebase* and the help from *ChatSDK*, the team thinks that it has been easier and more versatile working on the second iteration. It has challenged us for days, but we have been able to surpass everything.

In terms of convenience, *Firebase* shines, as its flexibility and its user-friendly console (*Dashboard)* and its wide documentation have made the job easier. Additionally, *Firebase* escalates easily relating to the number of users, and paying 200€ monthly is not an issue for a medium-big company.

It also makes any code adaptation for future iterations easier, as the database is just a *JSON* file which can easily be imported or exported.

Knowledge acquired

During this Final Degree Project, we have acquired a set of technological skills and company related knowledge.

In terms of used technologies:

- Development of JavaScript web applications. Although this programming language was taught on several occasions during the engineering degree, we have achieved certain neatness in the code, far superior to what was achieved in previous projects. Having used APIs such as *SendBird*, we have been able to observe professional programming patterns, and have obtained that mindset and applied it to the rest of the project. *Firebase* facilitated the comprehension of some object formats such as *JSON*, and we have especially acquired the capacity for programming secondary code, with a real time synchronous database.
- Development of mobile device applications. For this, the programming environment has been Android Studio, which some of our members had not much experience with and have been able to delve into it. We have also learned to solve many occurring issues during development with this environment.

In terms of the world of working with a company, Felipe has shown us how a company functions as well as new-coming technologies. He has shown us that the first path is not always definitive and that we should always explore more possibilities until getting to the best solution.

Future work

Although the process of developing the tool described in this Final Degree Project has concluded, not all is finished. The purpose of this section is to illustrate the lines of future work for this project.

Additional implementations

Due to lack of time, we could not implement the translations system with *Google Translation API*. To integrate this functionality, the code would have to be built on top of library *NodeJS*. Some other ideas for enhancing the chat would be:

- Implementing an unread messages **notifications** system for group chats. This is one of the *SendBird* features that have been impossible to add due to time restrictions. However, it is not a lengthy or expensive task. A *listener* would activate as soon as any of the group chats suffers a modification in the database within its messages, and then using its identifier, an unread messages counter

would be added to the rectangle of the chat room. A HTML element would be created and handled accordingly, showing or hiding that value.

- Adding group channel classifications by service subscriptions, such as spa or activities. The internal *Firebase* database structure would have to be modified to add that subscription field. Then, the program would run through every subscription in the group chat, creating and adding a new element to the HTML divider.
- Adapting *ChatSDK's* code to maintain coherence throughout the whole application due to these new implementations. New listeners would have to be added after deleting group chats or messages, as they are characteristics not present in the API. For this purpose, conferences would be held between *ChatSDK* creators and *Sismotur* and then contribute to the *GitHub* repository.
- Finish implementation for restoring the user's password. It is a functionality that has not had full attention, as it derives too far from the main use case.
- Adding a bot to the chat. It is featured in most well-known chats. It would save a lot of time to the receptionist in terms of answering frequently asked questions and advertising.

The next step would be to adapt all the code to a greater scope, with the possibility of storing multiple hotels in the database. To do this, it is only necessary to modify the structure of *Firebase rules* and *Firebase database*, and adapt all queries to go through one more level of depth. When logging in, the application would ask for the hotel identifier, and all internal queries would run according to that identifier.

Deployment for testing purposes

The next phase would convey the chat system deployment throughout hotels. For this, the first thing needed would be a reduced number of hotels to start testing the chat.

This process would serve as a functionality test and to see what could be improved. It would cost nothing to the establishment, as both parts would earn perks.

Another thing that could be put to practice, is the use of *beacons*. As described in *Chapter II*, the idea is to place this device in each room to serve as an entrance to the chat between room and hotel. In other words, the client arrives to his room, connects to the *beacon* and would have his chat set up in an instant. It would be necessary to calibrate the intensity of these devices in order not to interfere with each other. A person should not be able to join other chat rooms.

Other alternatives to *beacons* are QR codes or NFC technology. QR are a more economic option to *beacons*, as you only need to place a piece of paper on the table of each room with the printed QR. NFC offers a solution to the problem of the *beacon* being accessed in different rooms, having to place the phone closer than 20 centimeters away from the NFC.

Final deployment

Having concluded the previous phase and solved any occurring issues, the next step would be finally spreading the system to the maximum number of hotels.

For this, a good set of hotels for this to take place would be needed. *Sismotur*, being a large tourism company, would have no issue with earning a great number of clients.

If this system proves to work in Spain, a future vision would be spreading it throughout Europe and possibly the rest of the world.

Development in iOS

Another possibility for future work, would be the adaptation of *ChatSDK's* API for iOS. The initial idea for this Final Degree Project was developing *Inventrip* for iOS, but it was cancelled in the end due to some issues with a device that had a OS X version inferior to what was needed for developing in *Swift 3* with *Xcode*.

It could start with testing the Android app (which has a market share of 92,8%, approximately in Spain), and after analyzing results, proceed to the development in iOS.

Contribuciones

Enrique Ituarte Martínez-Millán

Nada más empezar el Trabajo de Fin de Grado, se me otorgó la tarea de poner a punto un Mac Mini prestado por nuestro director, Juan Antonio. En un principio, como íbamos a realizar un proyecto de iOS, necesitábamos poder programar en Swift 3 con Xcode. Sin embargo, este ordenador no tenía la última versión del sistema operativo, por lo que lo actualicé. Por desgracia, el ordenador posee poca memoria RAM, así que lo llevé al Apple Store para ver si le podían poner una versión compatible con Xcode 7. Esta tarea fue imposible, aunque la solución fuese otorgarle más RAM al ordenador.

Por estas razones, al final se decidió hacer el proyecto en Android y tecnologías web. Nada más comenzar el proyecto, estuve investigando sobre las distintas tecnologías web sugeridas, como Cycle.js. Tras varias sesiones de Skype con Felipe Santi (CTO de *Sismotur*), decidimos comenzar con el caso de uso del chat del hotel. Para ello, investigué sobre la tecnología *beacons Eddystone* y sugerí que se usase la API *SendBird* para lograr un *POC (Proof Of Concept)* sobre si sería viable un chat de la recepción de un hotel con las habitaciones.

En esta iteración del chat, colaboré con ideas de diseño y añadí algunas nuevas funcionalidades a la versión web, ya que no poseo los conocimientos de Android suficientes. Estas fueron, por ejemplo:

- Editar el nombre de un canal. Para ello, investigué sobre *XMLHttpRequest*, ya que *SendBird* podía realizar esta funcionalidad a través de una petición *PUT* al servidor.
- Eliminar un mensaje propio de un chat. Tuve que añadir y editar nuevas imágenes en formato *SVG*, investigar sobre esta funcionalidad en la mayoría de los chats más comunes, para dar con un diseño astuto.
- Diálogos modales de confirmación o toma de datos. Estos pueden ser al crear una nueva habitación (confirmación), o editar el nombre de un canal (entrada de datos).

También trabajé en tareas de revisión y adición de comentarios en el código. Traduje todos los textos de muestra de la API al castellano.

En la segunda iteración, investigué sobre el potencial de *Firebase* y configuré la aplicación web en la interfaz de la anterior iteración. Organicé todo el código para que nos resultase más cómodo adaptarlo a la nueva base de datos y repartí algunas tareas. Es aquí donde realicé mi mayor aportación al proyecto. Las funcionalidades importantes que realicé en esta etapa fueron:

- Carga de canales públicos en el desplegable *CANALES PÚBLICOS*, con los nombres de los mismos y las imágenes de perfil. Fue una tarea en un principio compleja al

no estar todavía familiarizado con *Firebase Database y Firebase Storage*. También conseguí añadir los canales públicos al contenedor, e implementé la nueva función para entrar en un canal público.

- Carga y configuración de los mensajes en chats. Esta parte fue quizás la más compleja, pero entre Delfín y yo conseguimos dar con la solución. Estuvimos la mayor parte del tiempo de la implementación con este problema, que describimos en la sección *Problemas encontrados*. Probé hasta creando funciones recursivas en donde cada llamada se hacía una consulta a *Firebase*, para que los resultados se fuesen añadiendo en orden al chat. La solución final, descrita en el último problema, la conseguí tras discutirla con Felipe Santi y Juan Antonio, y resultó ser bastante inteligente.
- Añadí la fecha de envío de los mensajes en los chats. Tuve que crear nuevas clases *CSS*, transformar la fecha en formato milisegundos a un formato legible, y las mostré al lado del nombre del autor del mensaje.
- Mostrar la lista de usuarios de un canal. Quizás de lo que menos me costó en todo el proyecto.
- Editar el nombre de un canal público. Adapté esta funcionalidad de la anterior iteración a *Firebase*, quitando el *XMLHttpRequest* anterior y sustituyéndolo con una actualización a la nueva base de datos.
- Añadí un filtro de usuarios en el desplegable de *NUEVA HABITACIÓN* y en el de *Lista de usuarios* del canal grupal, para que no resultase costoso encontrar los usuarios deseados. Tuve que diseñarlo de forma inteligente para que se aprovechase el espacio.
- Parametricé algunos diálogos modales y cree y diseñé nuevos, como el diálogo de difusión o el de creación de un nuevo canal público, donde añadí la posibilidad de subir una imagen de perfil. Añadí clases *CSS* para poder lograr el comportamiento deseado, quitándolas y añadiéndolas dinámicamente con *jQuery*.
- Investigué *Firebase Storage* para conseguir cargar imágenes en los mensajes y en las fotos de perfil de los canales grupales. Sobre este apartado, tuve un problema, y es que no se podían visualizar, debido al protocolo *file://*. Pero una vez cargados desde un servidor, se visualizarán correctamente.
- Cree un diálogo modal para el inicio de sesión, aprovechando y retocando la implementación de Delfín y Rodrigo. También añadí la gestión de errores, mostrándolos en pantalla.
- Cambié el diseño y los colores de algunos elementos, como el nombre de usuario en un mensaje, siempre que sea el usuario propio (se muestra en el color verdoso de *Inventrip*). Refiné el diseño final de la aplicación, para aprovechar espacio en el panel izquierdo.
- Añadí la funcionalidad de enviar mensajes de publicidad a todas las habitaciones a la vez. Para ello, añadí una imagen *SVG* en el contenedor *GRUPOS*. Cree una función que recorre todos los elementos del contenedor y llama a la que había implementado para enviar un mensaje individual.
- Solucioné algunos problemas de sesión que tuvimos, creando un flujo de inicio y carga de los elementos necesarios nada más iniciar sesión.

En cuanto a la redacción de esta memoria, mis aportaciones han sido:

- El resumen introductorio, escribiendo también el texto en inglés.
- El capítulo I, motivaciones, objetivos y estructura de la memoria.
- El capítulo II, donde hablo de *Sismotur*, su filosofía y las tecnologías punteras que están utilizando.
- Una parte del capítulo III, donde hablo *de HTML, CSS, JavaScript* y *jQuery*, además de *Firebase* como tecnologías utilizadas en el proyecto.
- El capítulo IV, donde hablo sobre la metodología de trabajo en este proyecto, el factor de forma, la postura del usuario, los métodos de entrada, y los elementos de datos en la aplicación web. También describo los principales requisitos funcionales de la aplicación. En la redacción de la segunda iteración, hablo un poco del diseño web y la mayoría de su implementación, en colaboración con Delfín.
- El capítulo V, donde describo la evaluación de ambas APIs, hablando de sus ventajas e inconvenientes y realizando una breve comparación entre ambas.
- En el capítulo VI, he escrito el resumen, he añadido la parte de implementaciones futuras en la sección *Trabajo futuro*, y he traducido todo el *Trabajo futuro*, *Competencias adquiridas* y el *Resumen* al inglés. También he descrito dos de los problemas encontrados y sus soluciones.
- En el capítulo VII, describo el anexo de la parte de *Firebase*.

También he realizado tareas periódicas de revisión a toda la memoria, corrigiendo errores de tipografía y revisando la redacción.

Ahora voy a tomarme el permiso de hablar sobre mis compañeros de trabajo.

Con ellos he trabajado en numerosas ocasiones, la más reciente en la asignatura Desarrollo de Sistemas Interactivos, donde conseguimos la nota más alta y aprendimos, sobre todo, a redactar. No puedo estar más orgulloso del trabajo que hemos realizado en este proyecto. Cada uno de nosotros ha sabido ponerse cuando otros flaqueaban, nos hemos inspirado, animado y apoyado durante todo el curso. Unos hemos hecho mucho de una parte y poco de otras, y la combinación ha sido muy buena.

También tengo que aclarar que, sin la ayuda constante de Felipe Santi, no podríamos haber hecho este proyecto de la misma manera. Gran parte del éxito se debe a su amplia disponibilidad y ganas de ayudarnos en cualquier cosa.

Delfín Álvaro Miguel Gómez

Al comenzar el Trabajo de Fin de Grado se nos encargó la tarea de investigar sobre diferentes herramientas para realizar la implementación de un chat. Tras investigar cada miembro del grupo las diferentes herramientas propuestas, se llegó a la decisión junto con Felipe Santi, CTO de *Sismotur*, de utilizar *SendBird*, una API muy bien preparada y con mucha documentación, perfecta para usarla en nuestra implementación.

Felipe también nos habló de diferentes tecnologías para el trabajo y la comunicación en grupo las cuales usaban en su empresa. Decidimos adaptarnos y utilizar estas tecnologías y estudiar su funcionamiento. Estas tecnologías estudiadas y aprendidas son:

- *GitHub*: nos sirvió para alojar nuestro proyecto y llevar un riguroso control de versión de la aplicación.
- *Slack*: es una herramienta de comunicación en equipo que utilizamos tanto para comunicarnos entre los miembros del grupo como con el CTO de Sismotur.

Una vez aprendidas estas tecnologías y decidida la herramienta a utilizar empezamos con la primera iteración.

En esta iteración del chat diseñé e implementé tanto en la parte web como en la parte Android. Mis principales aportaciones a la parte web fueron:

- Borrado de grupo: como la API JavaScript de *SendBird* no facilitaba esta opción, tuve que investigar sobre el protocolo *XMLHttpRequest* a través del cual *SendBird* permitía enviar una petición *DELETE* para borrar el grupo.
- Filtrado de habitaciones por hoteles: ya que en la base de datos de la aplicación se guardan las habitaciones de todos los hoteles y solo queríamos que cada hotel pudiera ver las suyas. Hubo que hacer algunas modificaciones tanto en la base de datos como en la parte web.

Mi principal aportación a en esta iteración fue en la parte de Android, ya que ya contaba con conocimientos sobre esta tecnología. Aquí mis aportaciones fueron:

- Implementación y diseño de la pantalla principal: aquí aparecen dos pestañas, cada una con una lista de chats. La primera contiene la lista de chats privados y la segunda la lista de chats abiertos. Cada chat lleva la imagen de grupo, nombre de grupo y número de usuarios.
- Implementación y diseño de la pantalla de chat privado. Aquí implemente diferentes funciones como:
    - Mostrar nombre de grupo.
    - Envió de mensajes de texto.
    - Recepción de mensajes de texto: mensaje con el nombre de usuario encima y la fecha y la hora a la derecha.
    - Lista de usuario del grupo: nombre y foto de cada usuario.

Añadir también que la API de *SendBird* da bastantes facilidades y documentación para el desarrollo de la aplicación Android.

En la segunda iteración estudie la documentación de *ChatSDK* para poder configurar la API y que así funcionase con nuestra cuenta de *Firebase*. Esto se hace a través de una API Key que proporciona la consola de *Firebase*, la cual ha de incluirse tanto en el SDK de *ChatSDK* como en la aplicación que desarrollamos. De esta manera, podían comunicarse nuestra aplicación Android con nuestra aplicación web.

Después de esta tarea investigué sobre el funcionamiento de *Firebase* en su documentación y recopilé algunos ejemplos de chats webs implementados sobre *Firebase* y proporcionados por la misma con el fin de que nos fuera más fácil la implementación de nuestra aplicación web. Dado que para el diseño de la parte web se reutilizó el de la primera iteración, en esta iteración la mayor carga de trabajo fue de implementación y solo hubo pequeños cambios en el diseño. En esta implementación, mis aportaciones fueron:

- Inicio de sesión y registro: los dos utilizan el método de correo y contraseña. Los métodos proporcionados por Firebase para estas dos funciones devuelven errores, los cuales Enrique se encargó de mostrar por pantalla.
- Recuperación de contraseña: esta función se encarga de enviar un email al correo del usuario para cambiar la contraseña en caso de haberla olvidado.
- Carga de chats privados: esta fue una de las partes más complejas de la implementación ya que no solo hay que cargar la lista de chats al iniciar sesión, sino que además hay que activar un *listener* por grupo que escuche cualquier cambio en el chat, como el añadido de otro miembro o la recepción de un mensaje y cuando se activara se actualizase la vista del chat. También había que añadir otro *listener* a la lista de chats del usuario por si otro usuario invitaba a un chat al usuario actual o se borraba un chat en el que estaba y así añadirlo o borrarlo de la lista de chats. Todo esto dio bastantes problemas al tener dos *listener* anidados, pero al final conseguí solucionarlo.
- Apertura de un chat privado: mostrar la pantalla de conversación, con el nombre de grupo como la lista de usuarios ordenados alfabéticamente, la barra de opciones de la conversación, la lista de mensajes y el cuadro de escritura de mensajes.
- Carga de lista de mensajes: en los chats privados la implementación era más sencilla que en la de chats abiertos ya que la lista de usuarios con sus nombres está contenida dentro de la propia conversación y simplemente había que añadir un *listener* para recibir los mensajes y mostrarlos en pantalla con el nombre de usuario al lado.
- Mostrar lista de miembros del chat: mostrar en un desplegable la lista de usuarios contenida en el objeto chat privado.
- Borrar chat privado: borrar un grupo que ha sido creado por el usuario actual.
- Invitación de usuarios al chat actual: mostrar en un desplegable la lista de usuarios que se pueden invitar al chat y incluir los seleccionados en el chat.

En la redacción de la memoria mi aportación fue:

- Tecnologías utilizadas, Chat SDK: donde describo esta API tanto sus funcionalidades como precios etc.
- Iteración I, diseño Chat SDK: donde explico y muestro las diferentes interfaces la aplicación.
- Iteración II, Implementación parte web: aquí explico parte de la implementación de la parte web en colaboración con Enrique
- Trabajo futuro:

- Despliegue en pruebas de la herramienta.
- Despliegue final de la herramienta.

Rodrigo Notario Pérez

Al comienzo del TFG, la empresa nos ofreció la posibilidad de elegir entre dos herramientas que tenían en mente para el desarrollo del sistema de chat que necesitaban: *SendBird* y *PubNub*. La primera función realizada fue una investigación de forma individual por cada miembro del grupo para comparar las dos opciones y ver qué nos ofrecía cada una. Posteriormente nos reunimos y pusimos las ideas en común para quedarnos con la mejor opción. Personalmente vi que *SendBird* estaba más enfocada a la herramienta que necesitaba la empresa y finalmente fue con la que nos quedamos, la cual era también la preferida por el CTO de *Sismotur*.

Una vez decidido esto, contribuí con el resto de mis compañeros al desarrollo del sistema de chat de la primera iteración utilizando la API de *SendBird*. Para ello, fue necesario leerme previamente la documentación disponible tanto de la versión Android, como la de JavaScript y la de la plataforma de la API.

En la parte web empecé mi aportación con:

- La implementación y diseño de la parte grupal de la versión web.
- También dediqué otra parte del tiempo al diseño general de la vista de la versión web, en el que realicé algunas subtareas como:
    - Creación de un diseño adaptado a *Inventrip*, con una apariencia minimalista aplicando los colores utilizados por la empresa.
    - Creación de imágenes en formato .PNG y .SVG.
    - Adición de botones, texturas y pulsaciones.

En la parte móvil, la mayoría de las tareas de implementación que realicé estuvieron relacionadas con la adición de funcionalidades al sistema de chat y su diseño. Entre ellas, están las siguientes:

- Implementar una funcionalidad al chat para poder enviar, recibir y visualizar en el *Smartphone* archivos en formato PDF
- Implementar una funcionalidad al chat para poder enviar, recibir y visualizar en el *Smartphone* imágenes en formato png, jpg, jpeg, gif, bmp y webp
- Implementar una funcionalidad al chat para poder enviar, recibir y visualizar en el *Smartphone* vídeos en formato mp4, 3gp, webm y x-matroska
- Diseño de las conversaciones tanto en grupos como en canales
- Diseño del icono de la aplicación siguiendo las líneas de diseño de la guía de *Material Design*

- Creación de imágenes para la apariencia de la aplicación.

Posteriormente, vino la segunda iteración de la que se ha hablado en la memoria.

En la versión web de esta iteración, contribuí de forma particular en:

- Implementación y diseño de una demo de un chat con *Firebase*. Para ello, manejé las siguientes herramientas:
    - Creación de un nuevo proyecto en *Firebase*
    - Manejo de varios tipos de autenticación tales como la autenticación mediante una cuenta de Google y configuración de la autorización
    - Manejo de node.js
    - Instalación de *Firebase Command Line Interface (CLI)* y configuración para su posterior utilización
    - Importación de *Firebase serve*
    - Creación y utilización de una base de datos JSON
    - Implementación del enviado de mensajes
    - Implementación de la sincronización de mensajes
    - Implementación de envío de imágenes en el chat
    - Implementación del almacenamiento de mensajes en la nube
    - Implementación de notificaciones de escritorio al recibir un mensaje nuevo en el chat
    - Activación de reglas de seguridad en la base de datos
    - Activación de reglas de seguridad en el almacenamiento
    - Despliegue de la aplicación utilizando un hosting
- Implementación de la búsqueda de usuarios
- Implementación de creación de grupo con un usuario
- Implementación y diseño de la página de registro y acceso por hoteles

En cuanto a la versión móvil, de forma individual trabajé en:

- Implementación y diseño de la pantalla de conversaciones de grupos
- Implementación y diseño de la pantalla de conversaciones de canales
- Implementación y diseño de la barra de herramientas
- Implementación y diseño de la ventana de inicio
- Diseño de la pantalla de canales
- Diseño de la pantalla de grupos

En cuanto a la memoria, mis contribuciones personales fueron:

- Estudio de la competencia. Para ello realicé el estudio tanto conversando con el CTO de *Sismotur* como realizando una investigación en Internet. Este estudio se incluyó el análisis de las siguientes herramientas:
    - WhatsApp Whenever Wherever
    - Justrequest

        - Virtual Hotel
        - Slack
    - Elementos de datos en Android
    - Una parte de la implementación de la primera iteración
    - Diseño web de la segunda iteración
    - Una parte de diseño móvil de la segunda iteración
    - Una parte de problemas encontrados y sus soluciones, hablando de:
        - Temática del TFG
        - Incompatibilidad al programar para diferentes versiones de Android, el cual fue un problema que me encontré implementando en la primera iteración
        - Nuevos formatos de pantalla ultra panorámicos
    - Conclusiones: competencias adquiridas en el desarrollo móvil y sobre la empresa.
    - Trabajo futuro: desarrollo en iOS

En cuanto al resto de mis compañeros decir que me ha encantado trabajar con ellos, ha sido un proyecto muy interesante en el que no sólo hemos conocido nuevas tecnologías, sino que también el CTO de *Sismotur* nos ha enseñado grandes conocimientos sobre el mundo laboral ahora que vamos a finalizar la carrera. Todos ellos han trabajado fenomenal y han sido muy buenos compañeros.

David Mata Lorenzo

Inicialmente, como ya se ha comentado anteriormente, el proyecto iba a consistir en la realización de una aplicación para *iOS*. en esta primera instancia, procedí a la instalación del sistema operativo en una máquina virtual para realizar pruebas de rendimiento, que dieron unos resultados bastante deficientes, ya que el sistema operativo consumía demasiados recursos y se nos hacía muy difícil trabajar en esas condiciones.

Posteriormente, se nos propuso un cambio orientado a las tecnologías web, entre las que se incluían diferentes alternativas como *Pyramid*. Tuvimos que estudiar dichas alternativas para decidir cuál utilizar, a pesar de esto y en consenso entre mis compañeros, el tutor y el CTO de *Sismotur* (Felipe), llegamos al acuerdo de implementar un chat para conectar *Android* y Web, proponiéndonos un par de servicios como son *PubNub* y *SendBird*, de entre los cuales, y tras un breve estudio, nos decantamos por el segundo.

Tras el estudio de estas tecnologías, nos pusimos manos a la obra.

Mi principal aportación en cuanto a lo que desarrollo se refiere, se basa principalmente en la primera parte del proyecto, la parte de *SendBird*.

Mi primera tarea consistió en el estudio, descarga y adaptación de la API de *SendBird* al proyecto Android, así como la integración de la parte web con la parte Android, comprobando el funcionamiento entre ambas antes de comenzar a manipular el código. Para ello, tuve que crear una cuenta y proyecto nuevo con el servicio de *SendBird*, y así obtener la API de acceso a la base de datos y al *dashboard*. Tanto en este proceso como en el proceso de desarrollo, intercambié una serie de mensajes con el equipo de ventas de *SendBird*, en los cuales me comentabas posibles soluciones a la hora de implementar las funciones que requeríamos, así como los planes de la versión de pago que más se ajustaban a nuestras necesidades.

En cuanto a la implementación, tanto en la parte de *Android* como en la parte Web, fue necesario realizar una serie de cambios en el código para su correcto funcionamiento.

Tanto en la parte Web como en la parte *Android*:

- Instalación y creación del proyecto, tanto el web como el de *Android*
- Cambio del diseño de las vistas, adaptando los colores y formas, evitando así trabajar con el chat estándar de *SendBird*, utilizando un diseño similar al de *WhatsApp*.
- Creación de la página de inicio, la cual permitía cambiar de usuario entre Recepción, Spa o Restaurante, y poder ver el funcionamiento de la aplicación de una forma más sencilla.
- Modificación de la forma en la que *SendBird* mostraba la información, dado que, en muchas ocasiones, no se ajustaba a las necesidades de nuestro chat.

En la segunda iteración:

- Edición de la información del usuario.

En cuanto a la memoria, mis contribuciones personales fueron:

- Investigación sobre Inventrip.
- Estudio de *SendBird.*
- Implementación Web de la primera iteración.
- Diseño web de la primera iteración.
- Diseño móvil de la primera iteración.
- Problemas encontrados:
    - *SendBird*: Bases de datos limitadas.
    - *SendBird*: Cambio de sesión de un usuario.

En cuanto a la relación con mis compañeros, puedo decir que, en ningún momento hemos tenido problemas de "convivencia" alguno, si bien es cierto que en algunos momentos hemos tenido dificultades para sacar adelante el proyecto, siempre hemos sabido apoyarnos y trabajar más cuando otros no han podido.

Relacionado con el tutor y el coordinador por parte de *Sismotur* (Felipe), ha sido un placer compartir este trabajo con ellos y agradecer la paciencia y la disponibilidad que han tenido, ya que era algo complicado compaginar los horarios para concertar las reuniones semanales que hemos estado realizando a través de *Skype* con el CTO de la empresa.

Capítulo VII

ANEXO

Introducción

En este capítulo, se expondrá información técnica referente a las dos APIs utilizadas en el desarrollo del proyecto, mediante algunos ejemplos destacables.

SendBird

Users

Los usuarios están identificados con un ID único, y deben configurar su Nickname e imagen de perfil.

Los campos básicos necesarios para un usuario son:

```
{
  "user_id": "john123",
  "nickname": "Johnny",
  "profile_url": "https://sendbird.com/main/img/profiles/profile_05_512px.png",
  "issue_access_token": true
}
```

Open Channels

Un canal abierto es un chat público. En este tipo de canales, cualquier usuario puede entrar y participar en el chat sin necesidad de pedir permiso alguno.

```
{
  "name": "Main Lobby",
  "cover_url": "https://sendbird.com/main/img/cover/cover_02.jpg",
  "channel_url": "open_channel_1",
  "data": "",
  "operators": ["terry5", "elizabeth2"],
  "custom_type": "lobby"
}
```

Group Channels

Un canal grupal es un chat privado, solo se podrá acceder a el con una invitación de un miembro perteneciente al chat.

```
{
  "name": "Chat with Lizzy",
  "cover_url": "https://sendbird.com/main/img/cover/cover_08.jpg",
  "custom_type": "personal",
  "data": "",
  "user_ids": ["terry5", "elizabeth2"],
  "is_distinct": true
}
```

Messages

Tenemos la posibilidad de enviar 3 tipos de mensajes diferentes, mensajes de texto, archivos o mensajes de administrador, los mensajes de administrador se podrán enviar tanto desde la aplicación (si dispones de los permisos necesarios) como a través del *Dashboard* de *SendBird*.

- Mensajes de texto

```
{
  "message_type": "MESG",
  "message": "Hey, how are you doing?"
  "custom_type": "Custom Type"
  "data": "Custom Data",
}
```

- Archivos

```
  {
    "message_type": "FILE",
    "custom_type": "Custom Type"
    "data": "Custom Data",
  }
```

- Mensajes de administrador

```
{
  "message_type": "ADMM",
  "message": "Jason has entered the room.",
  "custom_type": "entrance_message",
  "data": "Custom Data",
}
```

Firebase

En *Firebase console* disponemos de dos archivos: *Rules* y *Database*. Estos archivos tienen la siguiente estructura:

Rules

En este archivo, se escriben los permisos de lectura y escritura para los distintos usuarios de la aplicación. La sintaxis es clara de entender, ya que es muy parecida a *JavaScript*.

Durante el desarrollo, se recomienda dejar los valores por defecto:

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

Esto significa, que cualquier usuario va a poder leer y escribir en cualquier parte de la jerarquía de la base de datos. Sin embargo, según finaliza el desarrollo de la aplicación, se recomienda reajustar este archivo, de manera que se eviten posibles infracciones de seguridad.

Un ejemplo de esta configuración, puede ser la siguiente:

```
"threads": {
        "$thread_id": {
          "messages": {
            "$message_id": {
```

```
        ".read": "root.child($root+'/users/'+auth.uid).exists()",
        ".write":"root.child($root+'/users/'+auth.uid).exists()",

        "user-firebase-id":{
          ".validate": "root.child($root+'/users/'+newData.val()).exists()"
        },
        "creator-enitity-id": {
          ".validate": "newData.val() == auth.uid"
        }
      }
    }
  }
}
```

Esto significa que la lectura y modificación del identificador de un mensaje puede ser realizada si existe el identificador del usuario, y este ha iniciado sesión. El usuario creador del mensaje tiene que existir en la jerarquía de usuarios de la aplicación (/users/).

Un ejemplo más extenso de las reglas de los canales, sería la siguiente:

```
"threads": {
    "$thread_id": {

      // El usuario tiene que estar registrado para poder leer o escribir
      ".read": "root.child($root+'/users/'+auth.uid).exists()",
      ".write": "root.child($root+'/users/'+auth.uid).exists()",
        // Si por ejemplo queremos restringir a que el usuario esté en el propio chat:
      //".read": "root.child($root+'/threads/'+$thread_id+'/users/'+auth.uid).exists()",
      //".write": "root.child($root+'/threads/'+$thread_id+'/users/'+auth.uid).exists()",

      "details": {

       //"creation-date": {
```

```
  // This can only be set on creation - the time must be the current time
//  ".validate": "data.val() == null && newData.val() == now"
//},

"last-message-added": {
  // La fecha del último mensaje solamente se puede actualizar al momento exacto
  ".validate": "newData.val() == now || data.val() == null"
},

"name": {
  ".validate": "data.val() == null"
},

"creator-entity-id": {
  // Solamente se puede modificar en el momento de creación del chat
  ".validate": "data.val() == null && newData.val() == auth.uid"
},

"type": {
  // Tipo de canal
  // 0x0 (0) – Canal grupal (privado)
  // 0x1 (1) – Canal público
  ".validate": "data.val() == null && (newData.val() == 0 || newData.val() == 1)"
},



},
```

```
"messages": {
// Solamente los miembros de un canal pueden leer y escribir mensajes
".read": "root.child($root+'/threads/'+$thread_id+'/users/'+auth.uid).exists()",
".write": "root.child($root+'/threads/'+$thread_id+'/users/'+auth.uid).exists()",

 "$message_id": {

  "date": {
 // La fecha del mensaje tiene que ser la actual y solamente se puede modificar en
 // el momento de creación
   ".validate": "data.val() == null && newData.val() == now"
  },

  "payload": {
   // El contenido del mensaje sólo se puede modificar en el momento de creación
   ".validate": "data.val() == null"
  },

  "type": {
  // El tipo de mensaje tiene que ser 0, 1 ó 2, y sólo puede modificarse en el
 //momento de creación
   ".validate": "data.val() == null && (newData.val() >=0 && newData.val() <= 6)"
  },

  "user-firebase-id": {
   // El identificador del usuario del mensaje tiene que ser el mismo que el del
//usuario
   ".validate": "data.val() == null && newData.val() == auth.uid"
  },
```

```
      "read": {


      }
    }
  },
```

Data

El archivo de datos tiene un formato *JSON*, y se puede exportar e importar cuando se necesite, o modificarse directamente desde *Firebase console*.

Un ejemplo del aspecto de este archivo desde la consola es el siguiente:

Figura 6.01. Estructura del JSON

Como se ve en la figura 6.0.1, la consola permite maximizar o minimizar el contenido de la base de datos, así como añadir, modificar, o eliminar a mano cualquiera de sus datos.

El archivo real, tiene el siguiente aspecto:

```
{
 "testRoot" : {
   "public-threads" : {
    "-Ki0zgz96_DT85zORzI-" : {
     "null" : ""
```

```
  }
},
"searchIndex" : {
  "HpgO6bdx0gYjLuIFEaADbDF3H743" : {
    "email" : "delfinam@ucm.es",
    "name" : "delfin",
    "phone" : "6536666666"
  }
},
"threads" : {
  "-Ki1-goFVofb3XD9XEkw" : {
    "details" : {
      "coverUrl" : "henry.jpg",
      "creation-date" : 1492534873564,
      "creator-firebase-id" : "2K24gRd6v7cBz2NwDtSO9awwIcp2",
      "last-message-added" : 1494867812031,
      "name" : "Madrid",
      "type" : 1
    },
    "messages" : {
      "-KiEwAg5ltudO1U2YmPG" : {
        "date" : 1492768569498,
        "payload" : "Hey",
        "type" : 0,
        "user-firebase-id" : "2K24gRd6v7cBz2NwDtSO9awwIcp2"
      }
      "-KkC38sv6CVvHZIHk2Us" : {
        "date" : 1494867812031,
        "payload" : "Probando",
```

```
        "type" : 0,
        "user-firebase-id" : "Y1ZxnTP700hC8m748l4qrsSgVsw1"
       }
      }
     }
     "users" : {
       "HpgO6bdx0gYjLuIFEaADbDF3H743" : {
        "name" : "Delfin"
       }
       "Y1ZxnTP700hC8m748l4qrsSgVsw1" : {
        "name" : "Hotel Exe"
       }
      }
     }
   },
   "users" : {
    "2K24gRd6v7cBz2NwDtSO9awwIcp2" : {
     "color" : "0.635294 0.552941 0.686275 1",
     "last-online" : 1496249825743,
     "meta" : {
      "email" : "rodrinot@ucm.es",
      "name" : "Rodriguin",
      "pictureURL" : "https://firebasestorage.googleapis.com/v0/b/invertrip-chat-2c5fe.appspot.com/o/files%2Fjlk1b5sa4nvua35pbih9hpqqge_image.jpg?alt=media&token=072b1b89-d0ba-4495-9c52-d44d5927e64e",
      "pictureURLThumbnail" : "https://firebasestorage.googleapis.com/v0/b/invertrip-chat-2c5fe.appspot.com/o/files%2Fqi8ippvls347v5o1lu9dsk3h19_thumbnail.jpg?alt=media&token=9433f442-285e-4d7f-a1b9-ee0590c0ff20",
      "version" : "0.9.6"
     },
```

```
      "online" : false,
      "threads" : {
       "-KiXHAk9R0znsQCAxuo2" : {
         "null" : ""
       }
      }
     }
    }
  }
}
```


Esta jerarquía ha sido simplificada, borrando la mayoría de los datos.

Bibliografía

(Fecha de último acceso: 05/06/2017)

[1] Ruta Ribera de Duero*:* https://www.youtube.com/watch?v=lVLSAYLhMfg

[2] *Sismotur*: http://sismotur.com/

[3] *SendBird*: https://sendbird.com/

[4] *Chat SDK*: http://chatsdk.co/

[5] *FireBase*: https://firebase.google.com/?hl=es-419

[6] *WhatsApp* en hoteles: http://www.lahotelista.net/whatsappatencionclientemarketinghoteles/

[7] *WhatsApp Whenever Wherever*: http://www.europapress.es/turismo/hoteles/noticia-barcelo-illetas-albatros-implanta-servicio-whatsapp-clientes-20140610131920.html

[8] Datos de turismo en España en 2016: http://www.minetad.gob.es/es-es/gabineteprensa/notasprensa/2017/documents/turespa%C3%B1a%20avance%20cierre%202016.pdf

[9] Mapa de uso de *WhatsApp*: https://elandroidelibre.elespanol.com/2015/05/mapa-de-la-mensajeria-instantanea-facebook-domina-pero-whatsapp-no-es-todo.html

[10] Aplicación *Justrequest*: https://justrequest.de/about-us

[11] Aplicación *Virtual Hotel*: http://www.virtualhotel.com/about-virtualhotel/

[12] *Slack* en hoteles: https://blog.base7booking.com/2015/12/set-up-slack-for-your-hotel/